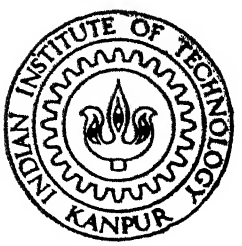
INTEGRATING KNOWLEDGE SOURCES IN DEVANAGARI TEXT RECOGNITION

by
VEENA BANSAL



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

DECEMBER 1997

INTEGRATING KNOWLEDGE SOURCES IN DEVANAGARI TEXT RECOGNITION

A Thesis Submitted

in Partial Fulfilment of the Requirements

for the Degree of

Doctor of Philosophy

by

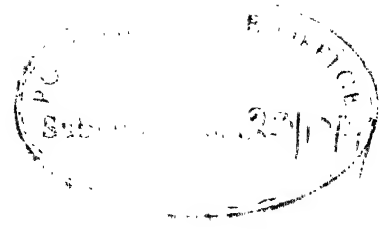
Veena Bansal

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

December, 1997



CERTIFICATE

Certified that the work contained in the thesis entitled "*Integrating Knowledge Sources in Devanagari Text Recognition*", by "*Veena Bansal*", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. R. M. K. Sinha)
Department of Computer Science & Engg.,
Indian Institute of Technology,
Kanpur.

Acknowledgments

I express my sincere gratitude for Prof. R.M.K. Sinha for introducing me to the fascinating area of pattern recognition and document processing. His constant support and encouragement have been available to me in spite of his unfavourable and rather demanding health condition during the initial stage of this work. It has indeed been a privilege to carry out this work under his supervision. Discussions with Prof. H. Karnick have been very helpful in clearing many of my doubts. This work was started as part of DOE sponsored project under the supervision of Prof. R. M. K. Sinha. The support is greatfully acknowledged.

I am thankful to Prof. Rajat Moona for taking care of my special lab requirements. I thank Prof. S. Biswas and Prof. A. Mukherjee for their keen interest and helpful suggestions. Prof. A. Jain and Dr. Renu Jain have been a source of stimulation throughout.

Veena Bansal

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A text recognition system has a variety of commercial and practical applications in reading forms, manuscripts and their archival etc. Such a system facilitates a keyboard less user-computer interaction. Also the text which is either printed or hand-written can be directly transferred to the machine. It is a great help to the visually handicapped when interfaced with a voice synthesizer. An elaborate list of the applications has been compiled by Govind [20]. The challenge of building a text recognition system which can match the human competing performance also provides a strong motivation for research in this field.

The human *reading* process is one of the most complex operations exhibiting human intelligence. There is general agreement among researchers in this field that the reading process consists of two major subprocess: *recognition* (the conversion of the written text into some language constituents) and *comprehension* (the organization of these forms into a meaningful conceptual entities which can be readily recalled). The recognition may be based on whole words, sub words, syllables, characters or a combination of these. The comprehension process has been widely studied [12, 38, 61] and various factors influencing the process have been pointed out. For example, the word frequency in the reader's lexicon plays an important role in the speed of the comprehension of the word [61]. If the word is more frequent, the processing

time for the word is lesser than if it is infrequent. Also, if a word is semantically very highly related to the preceding context, the processing time is further shortened [12]. As the interpretation of the text is constructed, a corresponding representation of the extensive meaning, of the thing being talked about, is also being built. If a reader cannot determine a referent, an attempt is made to determine the referential meaning by reading a larger chunk.

The automation of the process of recognition and comprehension has been a challenging problem for the researchers in the Artificial Intelligence field. Inspite of three decades of research, it has not been possible to build systems which could read arbitrary unconstrained text with human competing performance. One of the primary reasons for this failure is our inability to integrate variety of knowledge which humans use in the reading process into the machine reading process. In fact, the two phases of recognition and comprehension are not distinct and no clear boundary exists between the two. It is apparent from the fact that we take much longer time to read texts of non-sensical words/sentences.

A similar observation is made by Thomas and Bayer [6]. They observe that human typists have a significant increase in the error rate, if they do not understand the language the text was written in. Therefore, for a system intended to have only spelling capabilities, higher level of understanding is necessary in order to gain high accuracy on the letter level. In fact, the same is the underlying philosophy of the text recognition system described in this thesis.

A document may consist of text, graphs, pictures, tables etc. Text may be multi lingual and may contain mixed fonts. Various sizes of a font may also be used. Text could be hand-printed/written or machine-printed. A text recognition system extracts the text zone from the document and in most of the situations, is called upon to identify every letter/symbol of the extracted text zone. The identification process can be viewed as a case of the more general problem of pattern recognition. Gonzalez [19] has defined Pattern Recognition as *the categorization of input data into identifiable classes via the extraction of significant features or attributes of the*

data from a background of irrelevant details. For the character recognition problem, the input data consists of the images of a document and the identifiable classes are the character classes. The unit for recognition may be a letter/symbol or a word. The character recognition process may be aided by hypothesis generated by higher levels and/or may be post-processed for correction.

However, most of the commercial and practical system designs confine the role of reading process to that of substituting the keyboard entry to that of optical reading, and the application module to which this text is input, is developed without any integration with the OCR process. On the other hand, all natural language applications such as machine translation, question- answering, data-base retrieval etc. can provide this integration of the two phases. An example can be found in [3].

The name OCR (Optical Character Recognition) has been used in various context in the literature ranging from isolated character recognition to document reading systems. A system which deals with an unconstrained document is more appropriately called a document recognition system. The character classification phase after isolation of character boxes is often referred to as IOCR (Isolated Optical Character Recognition).

Both the tasks of isolation of character boxes and their recognition pose difficulties in real life situations. Some of researchers have taken a gestalt view of the pattern and attempt to recognize/postulate at the word level. There are several problems encountered in processing a document. A document may be multicolumn, consisting of images etc. The text zone of the document needs to be extracted before the recognition of the text can take place. In printed text, a skew results in overlap between text lines when scanned horizontally making it difficult to extract text lines. The background noise can further complicate the situation. The ink spread results in character fusions and fading fragments a character. Both of these may cause inaccurate segmentation and consequently incorrect classification. There are additional problems faced for recognition of the hand-written documents. The variation in the size and shape of characters, orientation, fusions and fragmentations

are more prominent in case of hand-written characters. Since 1940, many approaches have been tried for constrained/unconstrained printed/hand-printed/written text recognition with limited success.

In OCR, there is a conflicting demand of classifying a large set of natural variants into a single class and at the same time discriminate between closely resembling patterns. It is obvious that a merely statistical classificatory approach will not succeed and a deeper study into the structure of the scripts is required. The last 50 years of research has clearly demonstrated that no single strategy is sufficient for dealing with the complexity of the problem. Moreover, the strategy cannot be same for reading texts of different scripts/languages. The script specific features must be taken into account while devising the classification and segmentation strategies. For instance, scripts of Brahmi family have a two dimensional composition of symbols and are alphabetic. The segmentation strategy required for these scripts/languages is not needed for Roman script.

In the next section, the major works reported in the literature are briefly described followed by the approach proposed in this work.

1.1 A Brief Review of Earlier Work

The strategy used for OCR can be broadly classified into three categories:
(a) Statistical Approach
(b) Syntactic Approach
(c) Hybrid Approach
In statistical approach, a pattern is represented as a vector: an ordered, fixed length list of numeric features. An attempt is made to capture orthogonal features which are capable of correctly partitioning the feature space such that each partitioned zone corresponds to an unique character class.

In structural or syntactic approach, a pattern is represented as a set of simpler

shapes: an unordered, variable length list of geometric features of mixed type. The simpler shapes include strokes, end points, loops and stroke relations. The features represent global and local properties of the characters.

In hybrid approach, these two approaches are combined at appropriate stages for representation of characters and utilizing them for classification of unknown characters.

In the following subsections, some of the major attempts have been outlined.

1.1.1 Template Matching and Correlation Techniques

Mori et al report in [51] that in 1929, Tausheck obtained a patent on OCR in Germany and this is the first conceived idea of an OCR. Their approach was, what is referred to as template matching in the literature. The template matching process can be roughly divided into two sub-processes, i.e. superimposing an input shape on a template and measuring the degree of coincidence between the input shape and the template. The template which matches most closely with the unknown provides recognition. A very sophisticated non-commercial OCR was built based on this approach in 1962 by RCA group [23]. The two-dimensional template matching is very sensitive to noise and difficult to adapt to a different font. A variation of template matching approach is to test only selected pixels and employ a decision tree for further analysis. Peephole method is one of the simplest method based on selected pixels matching approach [51]. In this approach, the main difficulty lies in selecting the invariant discriminating set of pixels for the alphabet. Moreover, from an Artificial Intelligence perspective, template matching has been ruled out as an explanation for human performance.

1.1.2 Features derived from the statistical distribution of points

This techniques is based on matching on feature planes or spaces which are distributed on an n-dimensional plane where n is the number of features. This approach is referred to as statistical or decision theoretic approach. Unlike template matching where an input character is directly compared with a standard set of stored prototypes, Many samples of a pattern are used for collecting statistics. This phase is known as the *training phase*. The objective is to expose the system to natural variants of a character. Recognition process uses this statistics for identifying an unknown character. The objective is to expose the system to natural variants of a character. The recognition process uses this statistics for partitioning the feature space. For instance, in the K-L expansion [51, 41], one of the first attempt in statistical feature extraction, orthogonal vectors are generated from a data set. For the vectors, the covariance matrix is constructed and its eigenvectors are solved which form the coordinates of the given pattern space. Initially, the correlation was pixel-based which led to large number of covariance matrices. This approach was further refined to the use of class-based correlation instead of pixel-based one which led to compact space size. However, this approach was very sensitive to noise and variation in stroke thickness. To make the approach tolerant to variation and noise, a tree structure was used for making a decision and multiple prototypes were stored for each class. The Fourier series expansions [45, 58, 85], Walsh [30, 78], Haar and Hadamard [95] series expansion have been used by researchers for classification. Fourier descriptor for a curve is based either on amplitude and phase of harmonics or on a complex function of a point moving along the boundary. The Fourier Descriptor is invariant with respect to position and scale but depends on the starting point of the boundary tracing. An experiment was conducted by Lai and Suen [45] in 1981 on a data set of 100,000 hand-printed alphanumeric characters using Fourier descriptors. Recognition rate obtained was about 81%. They employed the local boundary features for finer classification and achieved a recognition rate of 98.05%. This experiment laid the foundation for the use of multiple features for classification.

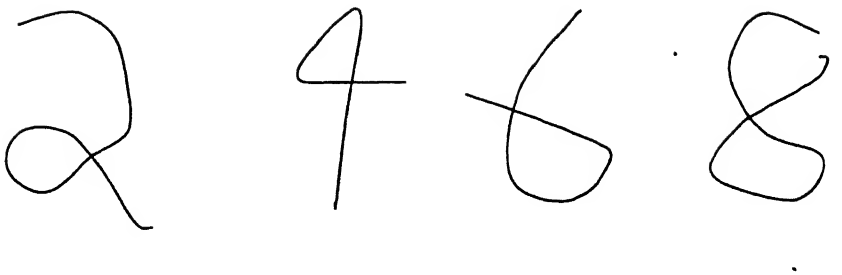


Figure 1: Characters which have two end points and one branch point.

Hough transform and projection transform [44, 43, 57] have also been used for classification. These features are computation intensive. Features derived from the statistical distribution of points include moments [1, 33] and crossings [5]. For a known font, the decision-theoretic approach has been found to be effective.

1.1.3 Geometrical and Topological Features

The classifier is expected to recognize the natural variants of a character but discriminate between similar looking characters such as O-Q, c-e, l-i etc. This is a contradicting requirement which makes the classification task challenging. The structural approach has the capability of meeting this requirement. For example the characters shown in figure 1 will belong to one class if only features selected are the number of end points and junction points. With additional features, these characters can be classified to four different classes.

The multiple prototypes [39, 46] are stored for each class, to take care of the natural variants of the character. However, a large number of prototypes for the same class are required to cover the natural variants when the prototypes are generated automatically. In contrast, the descriptions may be hand-crafted [64], and a suitable matching strategy incorporating expected variations is relied upon to yield the true class. The matching strategies include dynamic programming [97], test for isomorphism [77, 93], inexact matching [82], relaxation techniques [9] and multiple-to-one matching. Rocha et al [64] have used a conceptual model of variations and

noise along with multiple to one mapping.

Yet another class of structural approach is to use a phrase structured grammar for prototype descriptions and parse the unknown pattern syntactically using the grammar [17, 53, 54, 67, 69]. Here the terminal symbols of the grammar are the primitives of strokes and non-terminals represent the pattern-classes. The production rules give the spatial relationships of the constituent primitives.

1.1.4 Hybrid Approach

The statistical approach and structural approach both have their advantages and shortcomings. The statistical features are more tolerant to noise (provided the sample space over which training has been performed is representative and realistic) than structural descriptions. Whereas, the variation due to font or writing style can be more easily abstracted in structural descriptions. Two approaches are complimentary in terms of their strengths and have been combined [25]. The primitives have to be ultimately classified using a statistical approach. Baird has combined the approaches by mapping variable-length, unordered sets of geometrical shapes to fixed-length numerical vectors [4]. This approach, the hybrid approach, has been used for omni font, variable size character recognition systems [4, 39].

1.1.5 Neural Networks

In the beginning, character recognition was regarded as a problem which could be easily solved. But the problem turned out to be more challenging than the expectations of most of the researchers in this field. The challenge still exists and an unconstrained document recognition system matching human performance is still no where in the sight. The performance of a system deteriorates very rapidly with a deterioration in the quality of the input or with the introduction of new fonts/handwriting [64]. In other words, the systems do not adapt to the changed

environment easily. Training phase aims at exposing the system to a large number of fonts and their natural variants. The neural networks are based on the theory of learning from the known inputs. A back-propagation neural network is composed of several layers of interconnected elements. Each element computes an output which is a function of weighted sum of its inputs. The weights are modified until a desired output is obtained. The neural networks have been employed for character recognition with varying degree of success. The neural networks are employed for integrating the results of the classifiers by adjusting weights to obtain desired output. The main weakness of the systems based on neural networks is their poor capability for generality. There is always a chance of under-training or over-training the system. Besides this, a neural network does not provide structural description which is vital from artificial intelligence view point.

The neural network approach has solved the problem of character classification no more than the earlier described approaches. The recent research results call for the use of multiple features and intelligent ways of combining them [32]. The combination of potentially conflicting decisions by multiple classifiers should take advantage of the strength of the individual classifier, avoid their weaknesses and improve the classification accuracy. The intersection and union of decision regions are the two most obvious methods for classification combination [29].

1.1.6 On-Line Handwriting Recognition

One of the motivation behind the OCR/text recognition systems development has been to provide keyboard less interface to the computers. A direct entry by writing on a tablet is a natural way of communicating with a computer as compared to a keyboard or reading an already written page. The nature of the problem differs from that of off-line printed or hand-printed text recognition. The speed requirements are less stringent in case of on-line text recognition and has to match the speed of the human writer [91]. Average writing rates are 1.5-2.5 characters/second for English alpha numerals.

The temporal or dynamic information associated with writing is available to the system which helps in achieving better recognition rate. It has been shown that the on-line recognition is better than off-line on the same handwriting [7, 49]. Also, there is room for adaptation of a writer to the machine and a machine to the writer. For instance, the time between the end of a stroke to the beginning of next stroke characterizes a user and may be used for character segmentation. The early spatial segmentation algorithms were based on the projections on x-axis [18]. Recent spatial segmentation techniques check for a two-dimensional separation of the units [15]. The latest method combines spatial, temporal and other information to achieve word segmentation [15]. Acceleration and velocities of pen are used for eliminating wild points [56, 60]. Wild point elimination is part of the noise reduction phase which consists of smoothing, filtering, and wild point correction. Smoothing usually averages a point with its neighbours whereas filtering eliminates duplicate data points. Smoothing and filtering have been in use since 1966 [22, 91] and many techniques have been reported.

The classification of on-line characters based on dynamic information is more popular than static properties for obvious reasons. The features based on static properties are employed to reduce the set of candidate characters [40]. Dynamic information such as time sequence of zones, directions or extremes is used to represent a character [8, 21, 60]. A prototype dictionary is built and an exact match provides recognition. Table look-up becomes less practical with increased variation in characters. The sequences are compared by curve matching techniques. Curve matching is a popular technique for classification [34, 35, 56, 76, 98]. The curve is represented by points or their trajectory or both. Elastic curve matching is able to account for larger variation but is computationally intensive [90, 35, 98]. Another alternative is to use Fourier descriptors for representing the functions of time [2, 37, 36]. In recognition-by-generation or analysis-by-synthesis approach, the strokes of a character are mathematically modeled. This approach has been widely used for cursive script recognition. The recognition without segmentation avoids the segmentation problem but finds an application only where the number of words to be recognized is small. The internal segmentation has been tried along with analysis-by-synthesis

approach and has been found to work reasonably well [91]. Handwriting modeling has also been used in recognition [86] by finding characteristic invariant features.

The experimental systems have been reported to achieve a recognition rate of up to 98% on isolated characters. Results are almost same for constrained cursive scripts. The recognition rate varies from 60% to 90% for unconstrained cursive script when external segmentation is used [91].

1.1.7 Devanagari Text Recognition

Devanagari script is alphabetic in nature and the words are two dimensional composition of characters and symbols which makes it different from Roman and ideographic scripts. The algorithms which perform well for other scripts can be applied only after extensive preprocessing which makes simple adaptation ineffective. Therefore, the research work has to be done independently for Devanagari script. Some effort has been made in this direction mainly in India. A Sinha at al [50, 68, 71, 72] have reported various aspect of Devanagari script recognition. The post-processing system is based on contextual knowledge which checks the composition syntax. Sethi and Chatterjee [80] have described Devanagari numeral recognition based on the structural approach. The primitives used are horizontal line segment, vertical line segment, right slant and left slant. A decision tree is employed to perform the analysis based on the presence/absence of these primitives and their interconnection. A similar strategy was applied to constrained hand-printed Devanagari characters [79]. Neural network approach for isolated characters have also been reported [65]. However, none of these works have considered real-life documents consisting of character fusions and noisy environment.

1.2 Our Approach: Integration of Knowledge Sources

From the foregoing discussions, it is evident that knowledge plays a crucial role in human text recognition and that there is a need to integrate different knowledge sources in the machine reading system. The major challenge lies in taking the advantage of the strengths of a classifier such that its weaknesses are covered by other classifiers [29, 32]. This involves identification and representation of classification methods as well as of the environment in which each classifier performs optimally. An earlier attempt in integrating knowledge sources has been made by Srihari [87]. Rocha and Pavlidis have used hand-crafted labeled graphs for representing character prototypes and they call their approach knowledge based [64]. The ZIP code reading machines used by US postal department heavily rely on the context. The statistical structure knowledge for correction is also used [31, 62, 63, 83]. The contextual knowledge consisting of a word dictionary [24, 39, 89] has also been used for post-processing. Some researchers have attempts to use both the dictionary and statistical structural knowledge, at appropriate stages of processing [32, 72, 84, 87].

In this work, we have expanded the role of various knowledge sources in the context of Devanagari text recognition and attempted to integrate them in a meaningful way. Our document reading system is based on a hybrid approach for classification of characters and symbols where different knowledge sources have been utilized in an opportunistic manner. The character shape descriptors take into account the features that are distinct for the script (for example, holes in Roman as used in [4, 39]). This not only makes the description independent of size, font and style, but also facilitates use of these descriptions as filters to reduce the set of admissible characters over which further matching is attempted. Our descriptive schema for Devanagari characters is motivated with the above observations which we describe in chapter 5. The description generation and matching process are also described in the same chapter. We use three robust features as filters to prune the set of candidate characters. The first one is based on the coverage of the core strip, the

second on the presence/absence of a vertical bar and its position and the third one is based on the modified version of the horizontal zero crossings technique.

Next, the number of character boxes in each strip and the vertical bar property of the middle (core) strip character boxes are used to generate word envelop information. The word envelop information is used to select candidate words from a word dictionary which has been partitioned using word envelop in a hierarchical way. The hypothesis set for each character box is constituted by corresponding character of each selected word.

An intersection set of this hypothesis set and pruned candidate character set is constructed which is retained as the revised set of the candidate characters.

There is another aspect that we tend to ignore while devising methodology based on descriptions. The real life character images always tend to be fused or fragmented due to various reasons including noise. Any conceptual model (such as [64]), must be able to account for it. This does appear to be an easy task, as it is also dependent upon the segmentation and fusion algorithms incorporated into the recognition system. Therefore, we believe that in a practical system, training with real-life patterns is unavoidable. An automated trainer has also been designed and implemented which is described in chapter 7. The extraction of *characters and symbols* for recognition from the text zone is done in three steps. In first step, the text zone is segmented into text lines followed by text line segmentation into word. The words are then segmented into characters and symbols which is a two phase process (see chapter 4).

There are two distinct approaches reported in the literature for block segmentation: *smearing* [96] and *profiling* [52]. The *smearing* algorithm uses two thresholds, δx and δy for smearing in the horizontal and the vertical directions respectively. Different values of these parameters subsequently lead to block, line and word segmentation. However, these parameters cannot be dynamically adapted. As a consequence, the method is independent of the font size only up to a certain extent. The *profiling* method is a special case of Hough transform. The image is projected on a

vertical line. The projection along vertical line of a text blocks yields thick peaks corresponding to the dark pixels of the text lines separated by white gaps. Whereas, the graphics have a relatively uniform profile. The profile cuts are locally decided which makes the procedure font independent to a large extent. A projection of each text line on a horizontal line segments the line into words.

However, the overlapping text lines are not segmented into constituent text lines by either of the two methods. Some heuristics are used to further segment these lines [96]. In our implementation, we use the profiling method (see section 4.1 in chapter 4) with a two-pass mechanism to segment text zone into text lines including the overlapping text lines.

In Devanagari script, the characters and symbols are glued together with a horizontal line running on the top of the core characters. Therefore, the word boundary identification is easy and needs no special treatment. However, the segmentation of a word into the recognition units is involved due to the script composition. In the Roman printed text, most of the characters are vertically separate from their neighbours (apart from optional use of ligatures). However, at the stage of the digitization of the image, the local ink spread and aberrations, result in touching characters. The segmentation of the touching characters need special treatment. Many algorithms have been proposed for segmentation of touching characters [10, 39, 48, 64, 94]. All of these algorithms generate multiple segmentation points. Casey et al [10] select a suitable image part and performs template matching for classification. A different partitioning of the input pattern is attempted in case the classification fails. Liang [48] has proposed a recursive algorithm to segment the touching characters. The algorithm is based on two functions of the profile and projection of the touching characters. The algorithm, proposed by Tsujimoto et al [94], uses a logical pixel AND operation on two adjacent columns to estimate the segmentation cost at that column. All the columns above a preset threshold, become candidate cutting (break) points. The sequence of break points which results in classification of all the segments is accepted. This results in a large number of broken character errors such as 'm' getting segmented into 'n' and 'i'; 'h' into 'l' and

'i'. The knowledge about character composition (e.g. an 'm' is like a combination of an 'r' and an 'n') is used to merge the components during post processing phase. The success of the segmentation process depends upon the ability of the recognition process to recognize the components in different broken segments. Kahan et. al. [39] detect touching characters by using the ratio of the second-order difference of the vertical pixel projection to the value of the vertical projection as an objective function. The cutting points for touching characters are obtained in the horizontal positions where the segmenting objective function is maximum. This method is unable to separate highly fused characters.

In Devanagari script, a pure consonant (see appendix A) is deliberately made to touch the subsequent consonant as per the script composition conventions. Various modifiers are also attached to a consonant. These lead to a large number of touching characters. The nature of these fusions is different from the fusions caused by the local ink spread. Therefore, none of the above methods suffice for the segmentation of a Devanagari word into its constituent characters. A segmentation algorithm which uses structural properties of the script has been developed. This segmentation method has been described in section 4.6 of chapter 4.

The need for post-processing has been recognized by researchers in this field and many works in this direction have been reported. The correctly recognized characters need no further processing. However, a means for finding and correcting the classification errors is required. Many spelling correcting programs have been developed [59] which detect misspelled words and offer suggestions for the correct words. There are two approaches for judging the correctness. One estimates the likelihood of a spelling by its frequency of occurrence [31, 55, 62, 63, 83, 88] which is derived from the transition probabilities between characters. This requires a-priori statistical knowledge of the language. The other judges the correctness of the spelling by consulting the word dictionary [89]. Here, a mechanism is required to limit the search space. Number of strategies have been suggested for partitioning the dictionary [73] based on length of the word, envelop and selected characters. In practice, a combination of these strategies is used to ensure the selection of the right

partition in spite of certain classification errors [89]. Generally, dictionary methods yield higher performance in error correction as compared to the methods based on the transition probabilities. A hybrid [31, 83, 87] approach attempts to combine the best of both the approaches by amalgamating them at appropriate stage of processing. For Devanagari script, we use a two stage partitioning scheme based on the the word envelop and *tags*. A detailed description of the post-processing phase has been given in section 6.2 of chapter 6.

In this work, an attempt has been made to identify and integrate various relevant knowledge sources for Devanagari text reading system. External knowledge sources are acquired by a separate training phase. Whereas, the transient knowledge is extracted from the text as it is processed. The transient knowledge is made available to all other processing components of the system and is relevant only for that session/document. Every processing module makes use of the knowledge available. The knowledge sources have been put together with the help of the blackboard architecture [26]. This architecture facilitates use of many heterogeneous Knowledge Sources (KS) as well as heterogeneous ways of accessing them. Addition of a new knowledge source is also easy in this architecture. The contribution of each knowledge source is also visible to rest of the processes.

A brief introduction of Devanagari script is given in the appendix from OCR view point which is referred to in this work frequently.

1.3 Major Contributions and Achievements

The major contribution of this thesis work can be summarized as follows:

1. Development of a knowledge-integrated Devanagari Text Recognition Schema and its implementation.

2. Development of an effective algorithm for segmentation of touching printed Devanagari characters and its successful implementation.

3. Development of an algorithm for automated generation of description of Devanagari character shapes and its utilization in Devanagari character recognition.

4. Development and implementation of a process for Dictionary partitioning and search for words in Devanagari script for generation of hypotheses and their verification for their correction.

5. Development and implementation of automated training for generation of Devanagari character prototypes.

All the above work was started as part of a DOE sponsored project under the supervision of Prof. R.M.K. Sinha and several aspects of the above were investigated. However, refinement of these aspects, their integration, validation and testing has been carried out exclusively as this thesis work.

The complete system has been implemented except the pre-processing stage which includes text-zone extraction from the document. A recognition rate of above 85% at the character level and above 80% at the word level have been achieved on printed documents. We point out in section 8 that it can be further improved by integrating higher level of knowledge such as syntax and semantic knowledge.

1.4 Organization of the Thesis

Rest of the thesis is organized as follows. Chapter 2 describes various knowledge sources for Devanagari text reading system. Chapter 3 outlines design aspect and the architecture of the system. In chapter 4, segmentation process which extracts recognition units from the text has been described. The first phase of character

segmentation process is external while the second phase is invoked based on the outcome of the recognition process and information gathered by first segmentation phase. However, both phases have been described in the same chapter. The description schema, prototype generation and matching algorithms have been described in chapter 5. Post-processing phase has been presented in chapter 6. Chapter 7 gives the implementation details of our system along with experimental results. Chapter 8 presents the conclusion and directions for future work.

Chapter 2

Role of Knowledge Sources

Reading can be construed as the coordinated execution of a number of processing stages, such as word encoding, lexical access, assigning semantic roles and relating the information in a given sentence to previous sentence and previous knowledge [12]. The terms *bottom-up* and *top-down* has been used in literature to describe the two different reading theories [61]. In a bottom-up processing view of reading, lower level processes (e.g., letter and word recognition) are thought to occur prior to and independent of higher level processes. By contrast, in a top-down conception, reading is controlled by high level cognitive processes (e.g., making inferences), and lower level processes are called into play only as they are needed. However, there is informal agreement among researchers that lower level and higher level processes are both involved in reading.

It has long been known that better readers know more words than poorer readers. Vocabulary plays an important role in making a person a better reader. What reader knows about a topic also affects reading performance. Numerous studies have shown that prior knowledge plays a major role in reading text. If the children have greater knowledge than adults about a content area such as chess or dinosaurs, they perform better than the adults. The explanation for these findings is that subjects with a rich knowledge base can recognize more patterns and larger patterns than can

subjects with more improvised knowledge. Meta knowledge, the general knowledge about one's own selection and implementation of task-relevant skills, also affects the reading performance. One finding, for example, has been that young readers tend to regard reading more as an orthographic-verbal translation than as a meaning construction task. Sometimes, the barrier to comprehension may be in the encoding of the word itself [12].

In sum, humans use a number of knowledge sources for lower level processes as well as for higher level processes in reading comprehension. The lower level processes are responsible for moving the eye to the next input word and encoding the word into visual features [38]. The higher level processes are responsible for comprehension which involves determining the relations among words, the relations among clauses, and the relations among whole units of text.

In this chapter, an attempt has been made to identify various knowledge sources for a Devanagari text recognition system. The emphasis is on the lower level processes. Major challenge lies in creation, representation and invocation of appropriate knowledge sources.

A block schematic diagram showing various phases of a text reading system is given in figure 2. The system may make an error at any stage. The possible errors which we have been able to perceive for each phase have also been indicated in the same figure. At each stage, relevant knowledge sources help the system in detection and correction of these errors. The knowledge sources which are available before the recognition process begins, are referred to as external knowledge sources. External knowledge sources are acquired during a separate training phase or provided by the user. Structural properties of the script, statistical distribution of the symbols and classifying features are some of the external knowledge sources. Additional knowledge may be acquired during recognition process. This acquired knowledge is transient in nature and is meaningful only for the document from which it is collected. These knowledge sources are referred to transient or internal knowledge sources. The information about height and width of characters and symbols of a

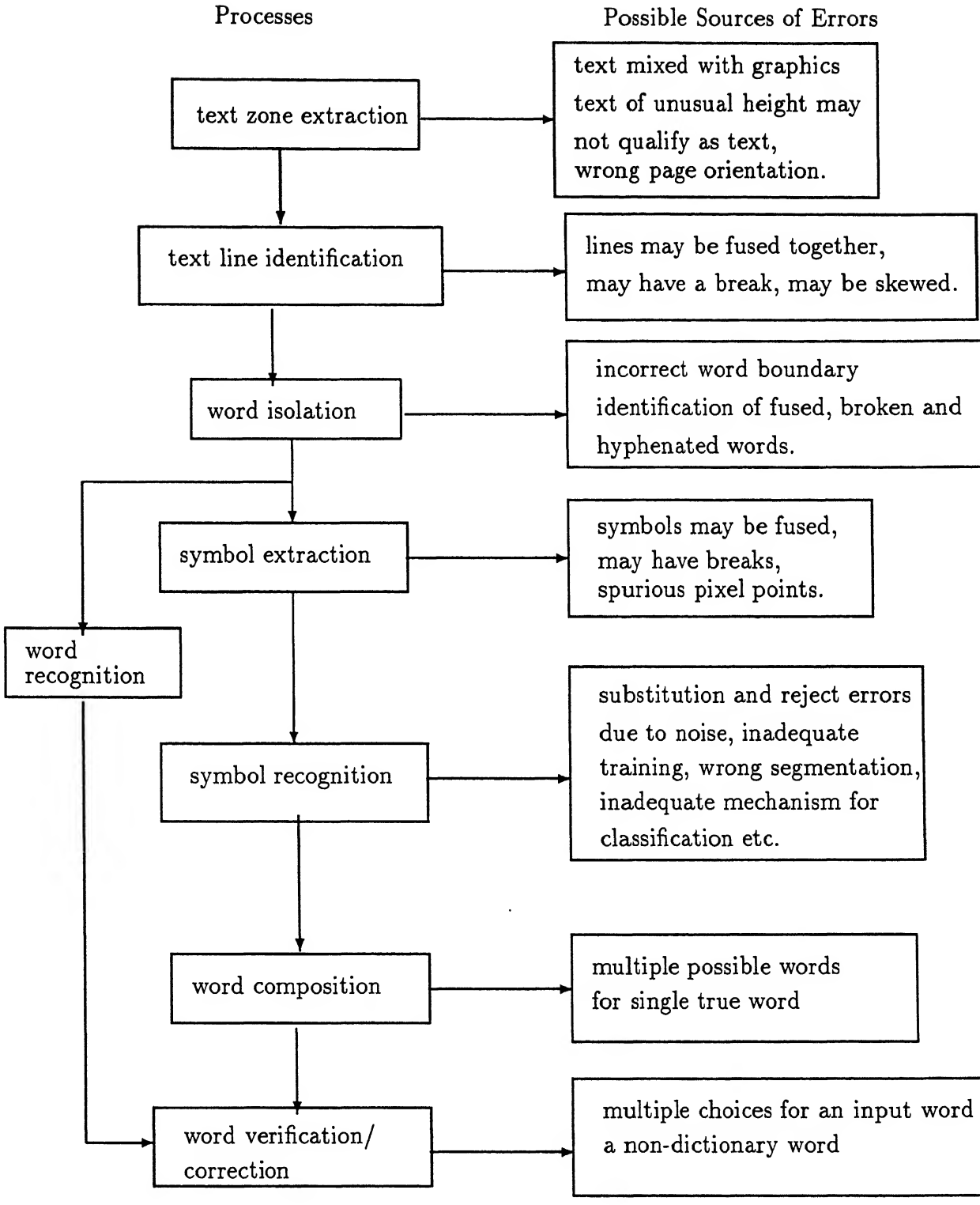


Figure 2: Phases of Devanagari Document Recognition Process and Possible Errors that each phase may make.

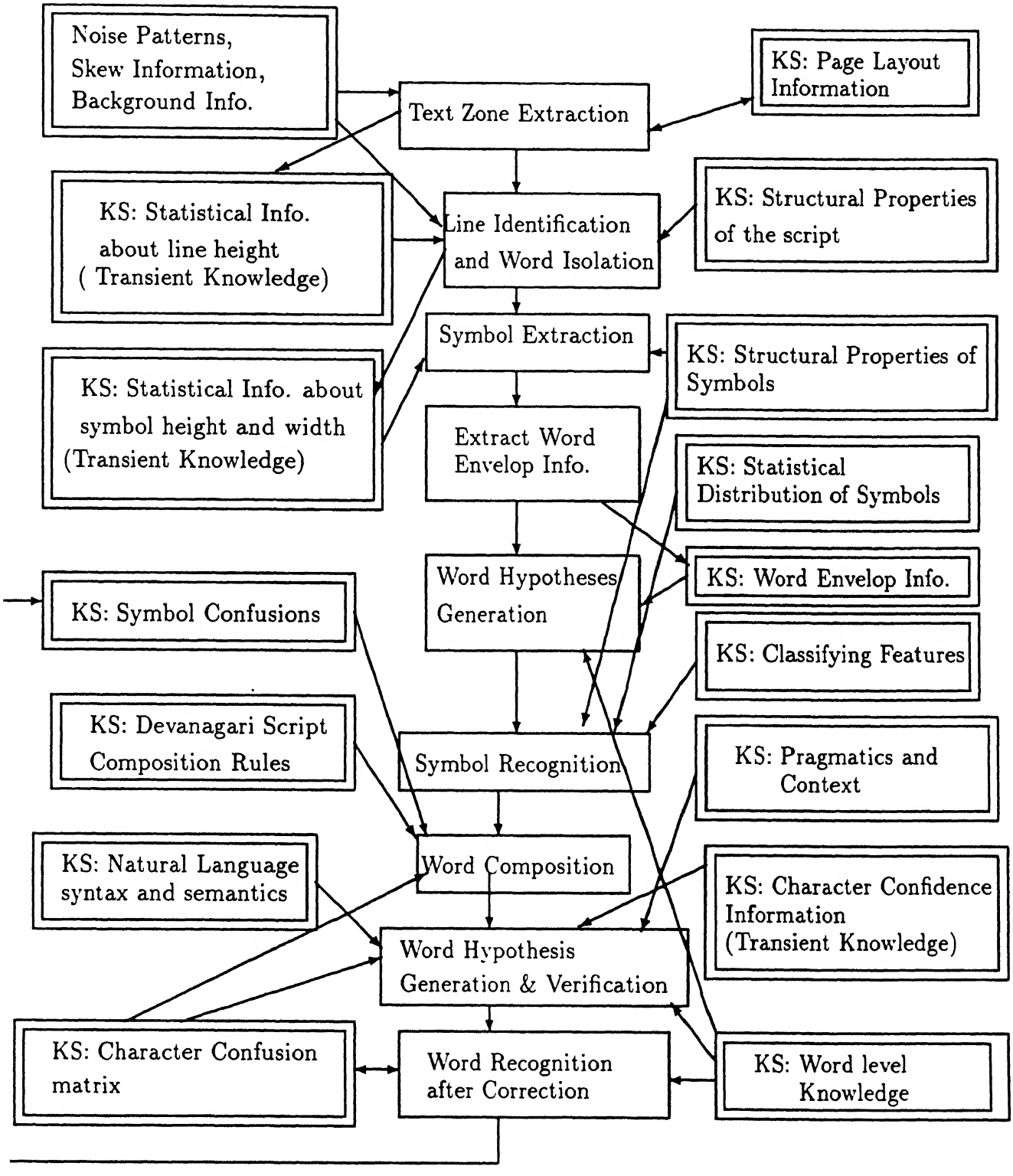
document is transient in nature and it can be acquired only during the processing of the document. Some of the major knowledge sources are shown in Figure 3 for the task of document reading. A knowledge kource may serve more than one phase. Each knowledge kource is elaborated further in the following sections.

2.1 Page Layout Knowledge

A document may contain text of different sizes, fonts and images. The documents may have different layout in terms of number of columns, placement of text and images etc. There are two widely used techniques for extracting text zone from a document, the Run-Length-Smearing Algorithm (RLSA) [96] and projection profile cuts [52]. Smearing means that all pixels between any two pixels are also set to be black, if they do not exceed a certain threshold. The smeared image in horizontal direction and vertical directions are combined by a logical OR operation. Height of all components of the combined smeared image are statistically analyzed. The height, around which most of the connected components cluster, is considered the height of text line. Connected components, within certain threshold of this height, are identified as text lines. A criticism has been that this method is font independent in a limited way [13]. This algorithm may suffice if the text lines are of approximately of the same height. In case, the document contains text of unusual height, such as titles, big headings or footnotes, this strategy may not work. Text lines of unusual heights may not qualify as text lines.

The projection profile method utilizes the nested structure of document information. The profile for text blocks are characterized by ranges of thick black peaks, separated by white gaps. Graphics, by contrast have a relatively uniform profile. This method is largely independent of font size as no preset threshold value is used.

However, if the background is not white or the text is surrounded by images or lines, the text zone extraction based on the methods described above may fail [10]. A solution is to provide a data base of various document page layout to the text



ure 3: Relevant Knowledge Sources for Devanagari Document Reading System up to word level;
s are shown in double rectangular boxes.

zone extraction module. This data base is used to identify the page layout of the document [10, 6]. Then, the document is searched in a predictive manner for the text zone. The document layout knowledge is an external knowledge source. Many documents are used to create a database of possible page layouts. An example page layout is shown in figure 4. This layout is used as a model layout to extract text zones such as title, author etc. from the text page shown in figure 5.

2.2 Background and Skew Information

The background of the document plays an important role in extraction of the text zone from the document. Similarly, document skew must be measured to ensure layout structure recognition accuracy. In the skew measurement process, the document image is divided into several equal-sized document wide swaths that are normal to the printing direction. Skew normalization is carried out by rotating the image using an affine transform operation and the original document is converted into a normalized image.

2.3 Height of Text Lines: Transient Knowledge

Each extracted uniform height text zone is segmented into text lines. Preliminary segmentation of the text zone is done assuming that no overlap or fusion occurs between text lines. This segmentation is based on horizontal histogram of the text zone being segmented. Height information of segmented text lines is analyzed and most frequent line height becomes the transient knowledge of height for the text zone under consideration. This height is referred to as *threshold line height*. This information is relevant only for the current text zone. The line segmentation module uses *threshold line height* for detecting possible overlapping text lines.

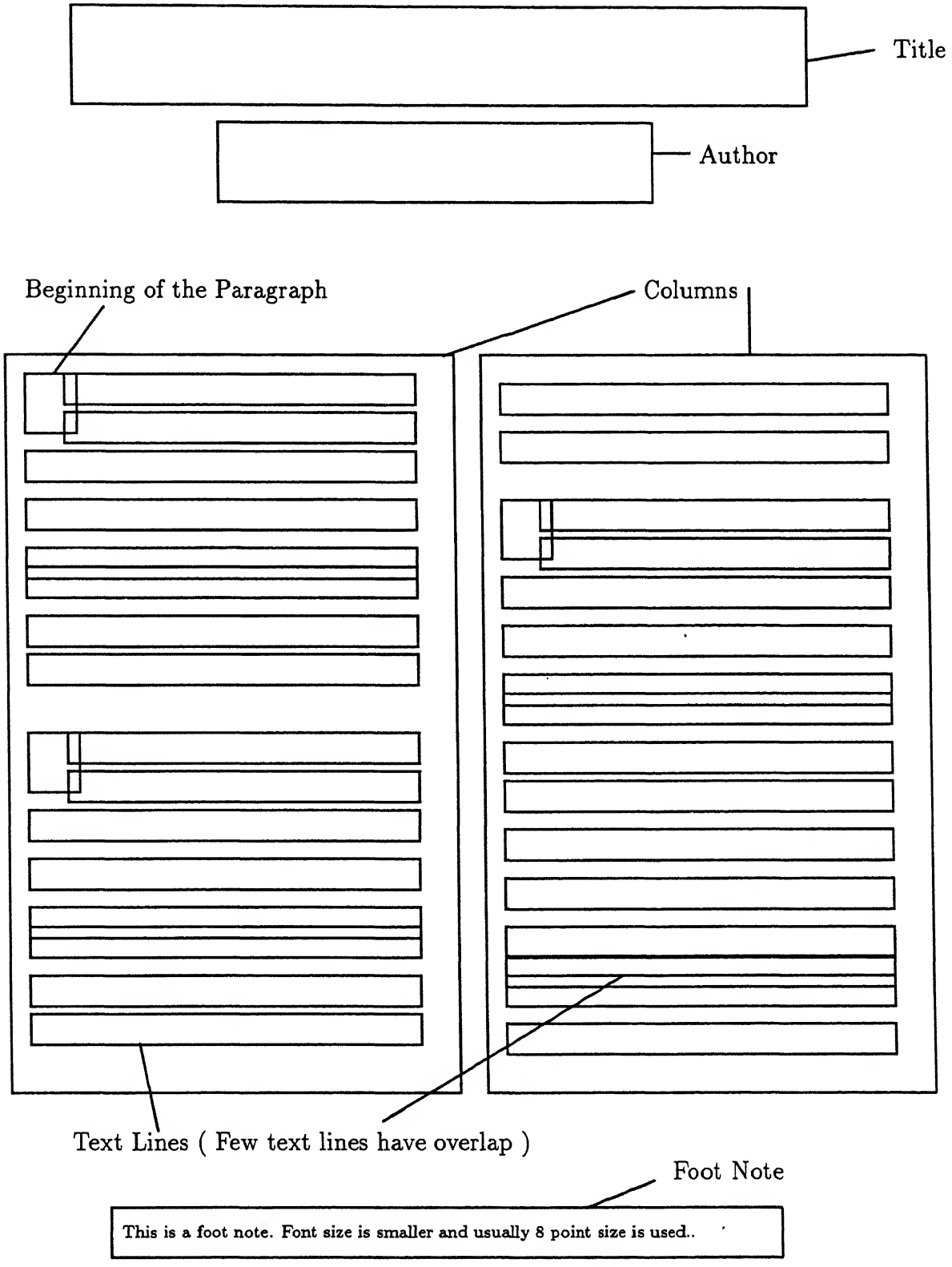
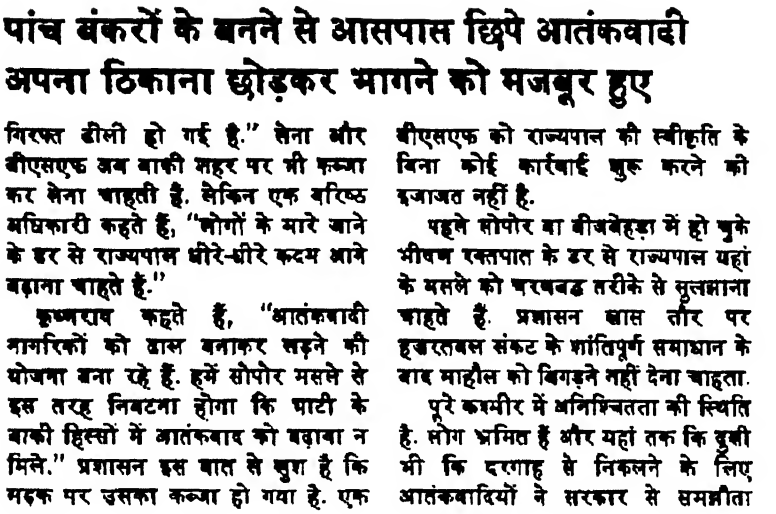


Figure 4: A Sample Document Layout that is used as a reference for extracting Uniform Height Text Zones from a document.



पांच बंकरों के बनने से आसपास छिपे आतंकवादी अपना ठिकाना छोड़कर भागने को मजबूर हुए

गिरफ्त ढीली हो गई है." सेना और बीएसएफ अब बाकी शहर पर भी कब्जा कर लेना चाहती है. लेकिन एक वरिष्ठ अधिकारी कहते हैं, "लोगों के मारे जाने के डर से राज्यपाल धीरे-धीरे कदम आगे बढ़ाना चाहते हैं."

कृष्णराव कहते हैं, "आतंकवादी नागरिकों को ढाल बनाकर लड़ने की योजना बना रहे हैं. हमें सोपोर मसले से इस तरह निबटना होगा कि घाटी के बाकी हिस्सों में आतंकवाद को बढ़ावा न मिले." प्रशासन इस बात से खुश है कि सड़क पर उसका कब्जा हो गया है. एक बीएसएफ को राज्यपाल की स्वीकृति के बिना कोई कार्रवाई शुरू करने की इजाजत नहीं है.

पहले सोपोर या बीजबेहड़ा में हो चुके भीषण रक्तपात के डर से राज्यपाल यहां के मसले को चरखबद्ध तरीके से सुलझाना चाहते हैं. प्रशासन खास तौर पर हजरतबल संकट के शांतिपूर्ण समाधान के बाद माहौल को बिगड़ने नहीं देना चाहता.

पूरे कश्मीर में अनिश्चितता की स्थिति है. लोग भ्रमित हैं और यहां तक कि खुशी भी कि दरगाह से निकलने के लिए आतंकवादियों ने सरकार से समझौता

Figure 5: A sample document that is segmented into uniform height text zones with the help of sample document layout shown in figure 4.

Text lines which are higher than the *threshold line height* for that zone are checked for a possible fusion. The text lines whose height is less than the *threshold line height* are checked for break in horizontal direction. Nature of overlap is script dependent. In Devanagari script, the overlap occurs due to lower modifiers of one text line and top modifiers of the following text line.

2.4 Structural Properties of the Script

In this section, the structural properties of the script are described which are used for line segmentation and word segmentation.

2.4.1 Header Line

All characters of a word are glued together by a horizontal line which runs at the top of core characters (see appendix A). This header line is the most dominating horizontal line in a text line. The text lines obtained by using the white space for text zone segmentation, may be fused or only a part of a text line. The presence of more than one header line indicates the fusion of two text lines. Similarly, the absence of a header line indicates that the image is part of a text line.

2.4.2 Three Strips of the Word

It is convenient to visualize a Devanagari word in terms of three strips: a core strip, a top strip and a bottom strip. Top and bottom strips have only modifiers and diacritic marks whereas the core strip contains all the characters and the vowel modifier ' ा' (see figure 72(a, b, c) of appendix A). This knowledge is used for extracting top modifiers, core characters and lower modifiers. The top and bottom strip may be empty for a word, only the top may be present or just the bottom. The core and top strip are separated by shirorekha. But no corresponding feature separates the lower strip from the core strip.

2.5 Statistical Information about Height and Width of Characters: Transient Knowledge

A text line is segmented into words. In Devanagari, identification of word boundaries is easy as all core characters of a word are glued together by header line. The words are segmented into *symbols* and characters. The horizontal and vertical histograms form the main basis for initial segmentation of a word into characters and symbols. The horizontal histogram is used to separate the top strip from rest of the word

by locating the header line. The top strip and the remaining word is segmented into characters and symbols which are vertically separate from their neighbours. Information about height and width of the characters of a text line obtained after initial segmentation is statistically analyzed. All characters are divided into three bins based on their height. The height corresponding to the bin which has maximum number of characters is stored as *threshold character height* which is taken as the transient core-strip height. Similarly, threshold width is also collected and stored as *threshold character width.* The height and width information is collected for each uniform text zone.

Some of the image boxes obtained after initial segmentation may require further segmentation. An image containing a compound consonant (also referred to as conjunct) needs further segmentation in vertical direction. A character with lower modifier needs segmentation in horizontal direction. All characters taller than the *threshold character height* are the candidate characters for further segmentation for extraction of possible lower modifiers. All characters wider than the *threshold character width* are the candidate characters for further segmentation for extraction of constituent characters from possible compound character boxes.

2.6 Structural Properties of the Characters and Symbols

In this section, structural properties of the characters and symbols are discussed. These properties are font independent to a large extent.

2.6.1 Structural Properties of Characters obtained by Visual Inspection

Character set of Devanagari script is divided into three groups based on the coverage of the region of the core strip. The characters which cover most of the core region are referred to as FULL BOX characters. The characters which cover upper region of the core strip are referred to as UPPER HALF BOX characters. LOWER HALF BOX characters are the characters which cover lower region of the core strip. These sets are shown in figure 6(a).

The FULL BOX characters are further divided into three groups based on the presence and position of the vertical bar. These three groups are: characters which do not have a bar: NO BAR characters; characters which have a bar in the middle: MID BAR characters; characters which have a bar at the end: END BAR characters. These sets are shown in figure 6(b). The END BAR set of core characters is large. These characters are subdivided in two groups based on the joining pattern of the character with the header line. The characters which touch the header line only at one point are put in one group and remaining are put in the other. These two classes are shown in figure 6(c). Classification based on these features is robust and remains consistent over a large number of fonts and sizes.

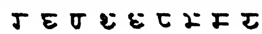
2.6.2 Joining Patterns for Conjuncts

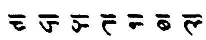
In a conjunct, one of the following joining pattern is observed depending upon the constituent characters of the conjunct:

Weak joining point Some of the half characters touch the following character very lightly as in the following conjuncts: न्त ज्य ज्थ त्त प्य म्म.

(a) Three classes of core characters based on the coverage of core strip

FULL BOX characters:
अ इ ई उ ऊ ए ऋ क ख ग घ ङ च छ ज झ ञ ट ठ ड ढ ढ त थ द ध न प फ ब भ
म य र ल व श ष स ह ष्क ह्र फ श स

UPPER HALF BOX characters:


LOWER HALF BOX characters:


(b) Three classes of FULL BOX characters based on the presence and position of vertical bar

END BAR characters:
अ ख घ च ज झ ञ त थ ध न प ब भ म य ल व ष स

MIDDLE BAR characters:
ऋ क फ

NO BAR characters:
इ उ ऊ ए छ ट ठ द ड ढ द र ह श

(c) Two classes of END BAR characters based on the joining pattern of the character with the header line

Character forms more than one junction with the header line:
अ ख घ झ थ ध प य भ म य ष स

Character forms only one junction with the header line:
च ज ञ न ब त ल व

Figure 6: Visually Extracted Properties of Devanagari Characters.

Thick joining point Some half characters have almost same number of pixels at the joining point, to the left and to the right of the joining point. Some such conjuncts are the following: e.g. च्य ब्ल ध्य.

Full height half character Third type of joining pattern is formed by half characters that have same height as the full character of the conjunct. All the characters in this category have either a weak joining point or a sudden fall in the pixel strength followed by a jump in the pixel strength near the joining region. e.g. स्य क्य क्स ख्य

This knowledge helps the segmentation process in proper segmentation of touching characters. For the weakly joined characters, weakest point is a likely break point. But is case of heavily touching characters, a different strategy is used in deciding the break point. The break point region, for the conjuncts involving full height half characters, is somewhat to the right compared to other conjuncts. This is described in detail in chapter 4.

2.7 Classifying Features

In this section, the classifying features which are used for Devanagari characters are discussed.

2.7.1 Horizontal Zero Crossings

Image of a character is treated as an array of pixels. A black pixel is expressed by a 1 and a white pixel by 0. Tracing the whole array row by row, number of transitions from black pixel to white pixel is recorded for each row. Let V_i be the number of transitions for ith row.

The sequence $V_i, 1 <= i <= n$, where n is the pixel height of the character, is referred to as horizontal zero crossing sequence S. Sequence S for 4 samples of character ह is shown in table 1. The sequence S is unique for all four samples of the character ह which indicates that this feature is sensitive to the font and writing style. Bao-Chang et. al. [5] use the sequence S as a classification feature.

We have modified the feature to make it font-independent and noise resilient. We have divided the character into three horizontal segments of equal height. Each segment is represented by number of zero-transitions that is most frequent in the segment. We divide the sequence S into three sub-sequences of equal length, S1, S2, and S3. For each subsequence the most frequent number is stored in S1_Most, S2_Most, S3_Most. The feature vector consists of (S_Most, S1_Most, S2_Most, S3_Most). Further, the sequence S is analyzed and the most frequent number in the sequence is stored as S_Most. The modified feature vector for all four samples of ह is (1121).

sequence S for four samples of character ह

$$1111111222222211111 => 1_8 2_7 1_5$$

$$1111122122222211111 => 1_6 2_2 1_1 2_6 1_5$$

$$1111111122222211111 => 1_9 2_6 1_5$$

$$1111111112222221111 => 1_{10} 2_6 1_4$$

Table 1: Sequence S for Devanagari Character ह; i_j indicates that horizontal zero crossings is i for j consecutive pixel rows.

This feature is used as a filter to reduce the set of probable characters for an unknown character.

2.7.2 Moments

Two dimensional moments have been widely studied as a feature for character classification [1, 11, 33]. A 2-dimensional image is treated as a rectangular grid.

For a 2-dimensional digital image, the moments are described as

$M_{jk} = \sigma X^j Y^k$

where summation is taken over all black pixels and (X, Y) are coordinates of a pixel.

Raw Moments The 2-D (p+q)th order moments of the image whose distribution function is (x,y) can be written as

$$m_{pq} = \int^{x} x^p y^q \delta(x,y) dx dy$$

This definite integral can be approximated by the summation over complete grid.

$$m_{pq} = \sum_{x=0}^{m} \sum_{y=0}^{n} x^p y^q \delta(x,y)$$

where $\delta(x, y)$ is the grey value of the pixel at the point (x, y) which is either 0 or 1 in this case. These moments are called raw moments and are information preserving if a large set is used.

Invariant moments Moments have been studied by Hu [33] and others to make them invariant under different transformations like

- Translation
- Scaling
- Translation and scaling

Translation The moments can be taken around the centroid to make them invariant under translation. The translation invariant moments are defined as:

$$\mu_{pq} = \sum_{x=0}^{m} \sum_{y=0}^{n} (x - \bar{x})^p (y - \bar{y})^q \delta(x, y)$$

where $p, q \geq 0$; $\bar{x} = \frac{m_{10}}{m_{00}}$; $\bar{y} = \frac{m_{01}}{m_{00}}$

Scaling If the image is scaled equally in both X & Y directions say by a constant alpha, the scale invariant moments are defined as:

$$\theta_{pq} = \frac{m_{pq}}{m_{00}^{(\frac{(p+q)}{2}+1)}}$$

where $p + q \geq 1$; $\theta_{00} = 1$; $\theta_{01} = \theta_{10} = 0$;

Translation and Scaling The moments invariant under translation and scaling can be defined as

$$\theta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{(\frac{(p+q)}{2}+1)}}$$

where $p + q \geq 1$; $\theta_{00} = 1$; $\theta_{01} = \theta_{10} = 0$;

In most of the moment based character recognition systems [33], only moments up to an order of four (i.e. p+q = 4) have been considered because it has been observed that higher order moments are increasingly sensitive to noise [1]. However, for real life samples of Devanagari script, we use moments up to an order of two only. These moments form a vector. The cardinal distance between the moment vector for an unknown character and a known candidate character is computed. This distance is used for ranking the candidate characters.

The moment vectors change considerably with a change in the font. Table 2 shows moments for three characters for two different fonts. These moments are computed for more than 20 samples of each character obtained after the segmentation from the printed document. The variance for the samples is shown next to each moment value. The difference in the moment vector is easily seen from the table 2 for these two

fonts. Therefore, in a multi font environment, this features has limited application. For a known single font document, moment are effective. For a set of known fonts, feature libraries are made and stored. The library corresponding to the font of the document being processed is accessed and used.

Font I

char	M_{11}	var	M_{02}	var	M_{20}	var	M_{12}	var	M_{21}	var
प	25.41	6.66	7.30	4.11	2.17	0.62	17.09	6.43	-2.94	0.72
क	12.81	3.57	3.44	1.71	0.89	0.27	20.45	5.50	0.61	0.22
ऋ	12.90	0.58	3.85	1.21	1.19	0.05	17.71	2.32	0.25	0.09

Font II

प	12.40	3.63	6.26	1.66	-0.39	0.11	22.81	12.70	-1.08	3.0
क	9.93	3.19	2.86	0.82	0.39	0.24	19.41	5.72	0.51	0.20
ऋ	13.17	6.62	3.05	6.09	0.47	1.51	21.43	6.41	0.20	0.09

Table 2: Moments for three characters for two different fonts; var stands for variance.

2.7.3 Aspect Ratio

Aspect ratio is the ratio of the height and the width of a character. This feature is easy to compute and divides the Devanagari character set into three or four clusters. The clusters have some overlap. This feature is font dependent and is used only for known fonts.

2.7.4 Pixel Density in 9-Zones

A character box is divided into 9 (3*3) zones (see figure 7). Total number of pixels in a zone is computed as:

(zone_height * vertical resolution) * (zone_width * horizontal resolution)
Number of dark pixels exceeding 50% of total pixels in a zone is represented by '1' in the feature vector. Less than 50% dark pixels correspond to a '0' in the feature vector. Each zone represented in terms of '0'/'1' constitute a feature vector. A feature vector (1, 1, 0, 0, 0, 1, 0, 0, 1) where the first value is of the zone number zero, followed by zone number one , two and so on, represents the upper modifier ` . For various patterns, some variation is found in the feature vector of a symbol. All distinct vectors for a character are stored as alternate prototypes. This feature is used for classification of lower and upper modifier symbols.

0	1	2
3	4	5
6	7	8

Figure 7: Division of a character box in 9-zones.

Various knowledge sources discussed in this chapter form an integral part of the document analysis and recognition process. We use blackboard architecture for integrating these knowledge sources which is described in the next chapter.

2.7.5 Number and Position of Vertex Points

The number of end points in a character image is also for classification. The position of these points is recorded in terms of 9(3*3) zones (see figure 7). There are more than one vectors for one character class due to variations in the characters. All distinct vectors obtained at the time of training are kept as alternative vectors for a character class.

2.7.6 Structural Descriptions of Characters

The description schema used here for the description of Devanagari characters consists of constituent strokes of the character and the relationship between them. The structural representation of Devanagari characters uses the slots which are shown in figure 8. The position of vertical bar is the first slot in the schema. The position of each stroke is stored in the schema in terms of the position of its begin and end points. The curvature and length information of each segment is also included in the schema. The matching process yields a distance figure between the structural representation of the known character and the prototype of the known character. This distance figure is used for ordering the candidate characters. This KS is described in detail in chapter 5.

Bar **Position**
Number of Strokes
Stroke 1:
  Position of Begin and End Points of the stroke
  Information of the Curvature of the stroke
  Information of the Length of the stroke
Stroke 2: ...

Figure 8: Description Schema for Structural Representation.

2.8 Character Confidence Information

The classification process applies a number of features to classify the unknown character image to one or more known character classes. For a feature, the distances from the reference prototypes for the candidate characters are computed. As the distance d, increases from the reference prototype, the chances of the unknown character belonging to the corresponding class decreases. The confidence figure c is

the term used for denoting the closeness of the feature of the unknown character to the reference prototype.

The confidence figure is either 0 or 1 for a feature of binary nature (match/no match); such as modified horizontal zero crossing vector, position of vertex points and position of vertical bar. Whereas, structural description comparison gives a distance from the reference prototype. The moments feature also yields a distance figure. The confidence figure can be inverse of the distance or any other monotonically decreasing function of the distance. In our implementation, the distance for moments, has been observed to vary between 0.0 and 50.00. The empirically decided maximum confidence figure is 1.0 corresponding to zero distance and for all other values of d, $c = 1/d$.

2.9 Character Confusion Matrix

There are certain characters which IOCR has difficulty in classifying to a unique known class due to their similarity with other characters. For instance, if the output of IOCR is अ, the true character could be any one from the set अ, ख, क्ष, भ. Similarly, if the output is ह्, the possibilities for true characters are ह्, ड्. The character confusion matrix is consulted when a substitution is made to map the IOCR output word to a dictionary word. The cost is less when the substitution is made by a known confusion than an arbitrary substitution. Figure 9 shows the output of IOCR and set of possible true characters.

2.10 Statistical Distribution of Characters

Frequency of occurrence of Devanagari characters has a wide variation. An analysis on 392 text pages from news magazines and story books containing 3,85,864 core characters was done. The statistical distribution of the characters according to the

Character Confusion Matrix for the output of IOCR:

IOCR output	Possible True Chars.	IOCR output	Possible True Chars.
अ	ख क्ष भ	ह्न	ह ड
उ	र	ऊ	रु
र	द	घ	अ ध
ज	न व ब	म	य प
ट	द ड	य	प म
त	ल	व	ब
ल	त	थ	य
न	व ब	प	य म भ
भ	म प	प	य म भ

Figure 9: Character Confusion Matrix for the output of IOCR.

analysis done, is shown in the table 3. The modifier symbol corresponding to vowel आ is the most frequent core symbol/character, constituting almost 30% of the text. It is followed by character क (8.31%). The characters in the table 3 are shown in the descending order of frequency of occurrence. The core characters are given first, followed by the characters in their pure form which are written as first character of a touching character pair. These characters constituted 1.75% of the total text. The characters in their rakar form are also given in the table; these constitute only 0.73% of the text.

2.11 Script Composition Rules

The segmentation process decomposes a word into core, lower and top strips. Each strip is further segmented into characters and symbols. A composition process is used to compose back the word from its constituent characters and symbols. A set of rules guide the process of composition [71]. These rules guide the composition processor in identifying the symbol sequences that are syntactically correct. These rules may be summarized as follows:

Char.	% usage	Char.	% usage	Char.	% usage	Char.	% usage
ा	29.82	क	8.31	र	5.41	न	4.72
ह	4.54	त	4.29	स	4.17	य	3.68
म	3.53	अ	2.54	व	2.53	ल	2.52
प	2.48	द	2.13	ज	1.97	ब	1.70
ग	1.57	भ	1.09	श	1.08	उ	1.01
ए	0.98	थ	0.97	इ	0.92	च	0.90
ध	0.81	ण	0.58	ट	0.56	ख	0.54
फ	0.30	ऋ	0.30	छ	0.27	क्ष	0.26
ष	0.20	ङ	0.19	ठ	0.17	ढ	0.12
घ	0.12	झ	0.11	ॠ	0.07	ऊ	0.03
ऩ	0.40	ॡ	0.27	ॻ	0.25	ठ	0.14
ऱ	0.12	॰	0.09	८	0.09	ॾ	0.08
६	0.08	ॲ	0.08	ज	0.07	रू	0.05
४	0.01	९	0.01	फ	0.01		
प्र	0.61	क्र	0.06	ऋ	0.02	ब्र	0.01
फ्र	0.01	म्र	0.01	भ्र	0.01	घ्र	0.00
ध्र	0.00	व्र	0.00	ख्र	0.01	स्र	0.00
ज्र	0.00	य्र	0.00				

Table 3: Devanagari characters in the descending order of their usage in the text consisting of 385864 characters comprising 39 documents.

- a vowel modifier symbol can appear only on a consonant;
- only one vowel modifier can be attached to a consonant;
- no modifier symbol or pure consonant can appear as a stand alone symbol, the only exception being a pure consonant in "halant" modifier form at the end of the word (mostly used in Sanskrit);
- a consonant cluster is treated as a consonant for composition purposes;
- the lower symbol ্ can appear only on a non-endbar consonant and "Rakar" occurs only with end-bar or middle-bar consonants.
- the symbol ঁ can appear in combination with or without any vowel modifier.
- any combination of diacritical marks may be used (although not all combinations are semantically meaningful on all characters).

This set of rules is used as a knowledge source by the composition processor.

2.12 Word Envelop Information

The number of character boxes in each strip and the vertical bar property of the middle (core) strip character boxes are used to generate word envelop information. The word envelop information is used to select candidate words from a word dictionary which has been partitioned using word envelop in a hierarchical way. The hypothesis set for each character box is constituted by corresponding character of each selected word. This knowledge kource is described in chapter 6 in detail.

2.13 Word Level Knowledge

Word level knowledge is represented as a word dictionary which may consist of a primary dictionary and an auxiliary dictionary. Auxiliary dictionary is used for storing proper nouns, acronyms etc.

There are certain character pairs which are visually similar and an IOCR is likely to confuse between them resulting in substitution errors (see figure 9). Words, which are not present in the dictionary, may contain some substitution errors (see figure 10). The characters which have large distance from the reference prototypes (in other words, have been recognized with low confidence) may be the substitution errors. These errors can be corrected by selecting alternative words from the dictionary in a constrained manner. These constraints come from the knowledge of the envelop of the input word. Width of the words and its constituent characters also play a crucial role in selection of the alternatives. It is also possible that the substitution errors lead to a valid dictionary word which is not the true word. Such errors can be detected only by using higher level of knowledge such as context, syntax and semantics etc..

Character level errors have changed 3rd and 5th word to non-dictionary word

input sentence: यहाँ हिन्दु पारसी ईसाई सभी रहते हैं

output sentence: यहाँ हिन्दु मारसी ईसाई समी रहते हैं

Figure 10: Examples from Devanagari script and Hindi Language.

It is also possible that the true word is not present in the dictionary. It happens in the case of proper nouns, acronyms etc.. Such words should not be forced to map onto dictionary words. If the word has been recognized with high degree of confidence and it is not present in the dictionary , it is left unchanged.

The word envelop information is useful in limiting the candidate words during the

dictionary search. The number and position of the top and lower modifiers decide the envelop of a word. In chapter 6, we describe a partitioning scheme for a Hindi dictionary. The search process and the strategy for selection of alternatives are also described in the same chapter.

2.14 Natural Language Syntax and Semantics

A word found in the dictionary is assumed to be the true word. However, there is a possibility that the input word is mapped to another dictionary word due to classification errors. The system may use additional information about each word such as its syntactic catagory, syntactic and semantic expectations for the preceding and following word(s)/phrase(s). This additional information can help the system to detect these errors if the syntactic category of the mapped word is different from the expectations generated by previous word. In case, syntactic category remains unchanged, other details like gender, person, number and class can be checked.

2.15 Pragmatics and Context

The errors which remain after the use of language syntax and semantics can be further eliminated by using the pragmatics and context. The words pertaining to a specific context such as medical science or poetry can be better dealt with by using context.

Chapter 3

System Architecture

A document reading system may be looked at as a hierarchical organization of cooperating modules. At the lowest level, the document reading problem consists of the document image. And at the top level, the interpretation of the input image exists in the form of recognized words and sentences. The segmentation and recognition phases lie between these two levels. What should be the strategy for invoking modules at various levels? There are three well known strategies which are listed below:

- go in a predefined sequence;
- the invocation of a module be based on an expectation/guess or based on the outcome of other stages;
- mixture of the above strategies.

Since the discriminating features of a character manifest in different ways in different situations, the recognition process does not always take the same path. Classification process needs to use different Knowledge Sources (KSs) for different characters. We need to ensure that application of one KS does not adversely affect application of other KSs. Therefore, an architecture is required which:

- provides freedom to the control structure to dynamically select a module and invoke it.
- facilitates addition/deletion of a knowledge source.
- supports heterogeneous representation of a KS along with heterogeneous ways of accessing them.

A blackboard model [14] for solving a problem fulfills these requirements. The blackboard model consists of two basic components:

The knowledge sources The knowledge needed to solve the problem consists of various knowledge sources, which are separate and independent.

The blackboard data structure The problem-solving state data are kept in a global database, the blackboard. Knowledge sources, when invoked, produce changes to the blackboard. These changes lead incrementally to a solution of the problem. Communication and interaction among the knowledge sources take place solely through the blackboard.

There is no control component specified in the blackboard model. The traditional control structures are either goal-driven or data-driven. The blackboard model has been used with opportunistic control mechanism which dynamically selects a KS and invokes it. The ease of interaction, visibility of contribution of KS and independence of knowledge sources in blackboard model provide an appropriate architecture for the control component as well. The way solution blackboard holds partial solutions and competing & cooperating processes work; the same way, different control plans that govern the solution generation are posted on a separate blackboard called control blackboard. Hayes-Roth [26] describes the behavioural goals which can be achieved by the control blackboard architecture. One of them is the freedom to use variable grain-size control heuristic. For a document reading system, the variable grain-size control heuristics correspond to reading a document at different levels. It is

conjectured that the human reading process progresses from coarse grain to fine grain-size. An initial look at a document is used to read legible words, possibly by just looking at the envelop of the word. Whereas, the difficult-to-read words are read with elaborate strategies. A partially read word is conjectured to be one of the words of a set which is decided based on the recognized characters of the word and other words of the text. The set of conjectured words may change as more words of the text are read. In sum, a seemingly right reading strategy at initial stage may loose its importance as the reading progresses. This corresponds to dropping out a control strategy from the blackboard if it becomes unproductive. Multiple control strategies may be active at the same time; some may become inactive as the partial solution is posted on the blackboard. The control plans are activated or deactivated from the control blackboard based on the partial solutions or event that occurs in a dynamic situation. An agent performs the job of choosing among possible actions from the control board and for the knowledge application [27] [28].

We have designed an architecture for Devanagari document reading system based on the blackboard model. We do not consider the on-line document reading and therefore, the control strategy need not worry about the dynamic situation. In the following sections, the architecture and its components are discussed.

3.1 System Architecture and its Components

The system architecture and its components are shown in figure 11. The system consists of a solution blackboard and various knowledge sources. There is a set of tools that is used for applying knowledge sources. Each component is discussed in the following sections.

We have a diverse set of knowledge sources incorporated in our system. The representation also varies according to the intended usage.

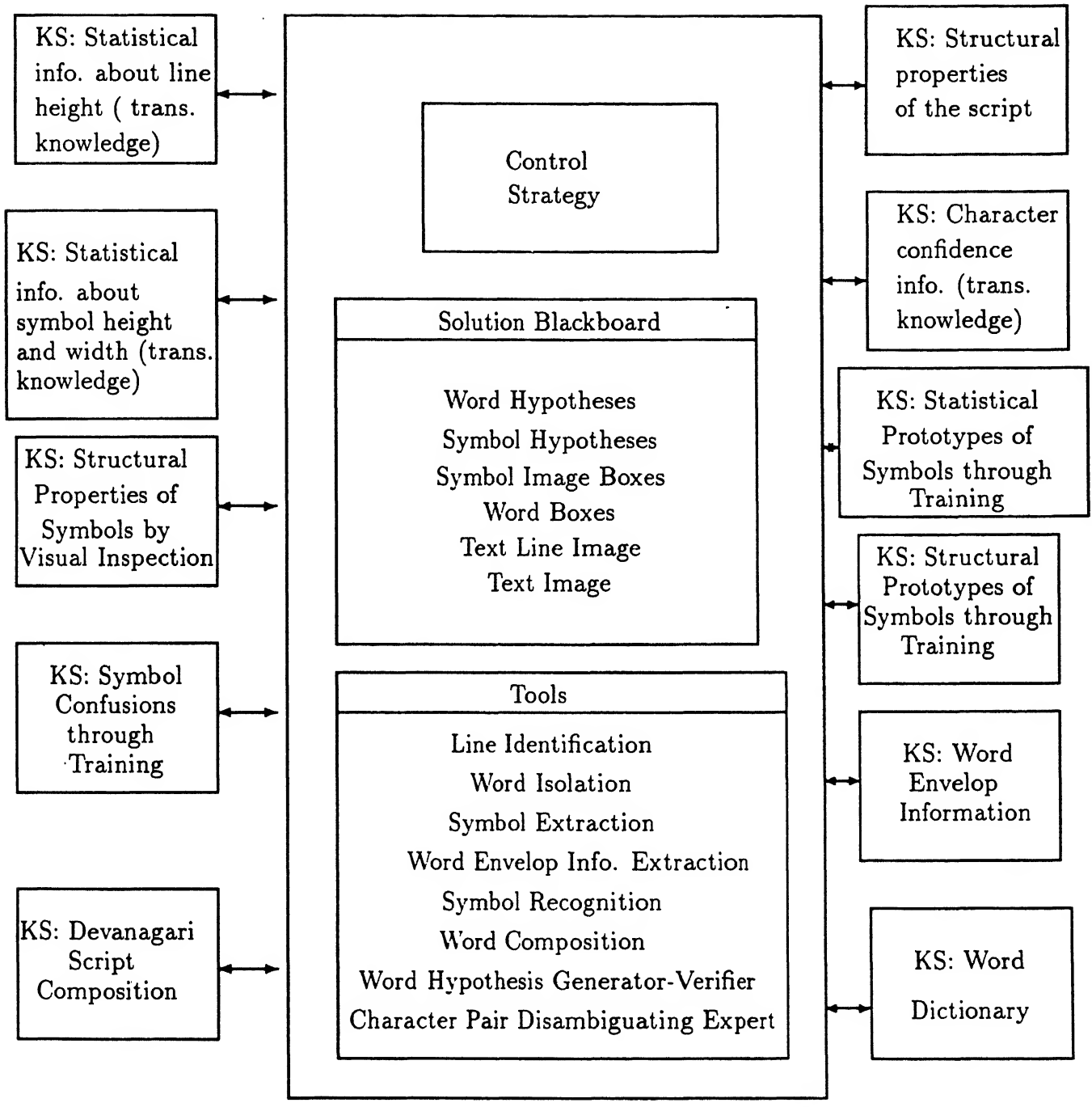


Figure 11: Blackboard Architecture For Devanagari Text Recognition System.

3.1.1 Statistical Information about Line Height: KS1

Statistical information about line height is analyzed and the most frequent line height is represented as *threshold line height* (see section 2.3 of chapter 2). It is a single integer number which represents height in pixel lines.

3.1.2 Structural Properties of the Script: KS2

Header line glues characters of a word together. The top modifiers are placed above the header line and lower modifiers are placed below the characters (see section 2.4 of chapter 2). This knowledge has been embedded in the processes which use this information and has not been stored explicitly.

3.1.3 Statistical Information about Height and Width of Characters: KS3

Height and width of characters are stored as *threshold character height* (KS3-1) and *width* (KS3-2); both in pixel unit. This knowledge is transient and generated internally (see section 2.5 of chapter 2).

3.1.4 Structural Properties of Characters obtained by Visual Inspection: KS4

There are three independent properties which constitute this KS.

- *Coverage of the Core Strip: KS4-1*

The character set is divided into 3 classes based on the coverage of the core strip (see section 2.6.1 of chapter 2). This knowledge is represented as three classes of characters: one for the upper half strip characters, one for the lower half strip characters and one for the remaining characters.

- *Position of the Vertical Bar: KS4-2*

The character set is divided into 3 classes based on the presence and position of the vertical bar (see section 2.6.1 of chapter 2). This knowledge is represented as three classes: one for end bar characters, one for middle bar characters and one for no bar characters.

- *Number of Junctions with Header Line: KS4-3*

The end bar characters are further divided into two classes based on the number of junction that a character forms with the header line. There are two classes: one for the characters which form one junction and the other for the remaining end bar characters (see section 2.6.1 of chapter 2).

- *Joining Patterns for Conjuncts: KS4-4*

There are three types of joining patterns which are observed in Devanagari conjuncts. This knowledge has been implicitly used by the conjunct segmentation process (see section 2.6.2 of chapter 2 and chapter 4).

3.1.5 Statistical Prototypes for Characters: KS5

Three independent KSs constitute KS5; Two of these are used for core characters; The third one is used for upper and lower modifiers.

- *Modified Horizontal Zero Crossings Vector: KS5-1*

The first one is the modified horizontal zero crossing vector which consists of four integer values (see section 2.7.1 of chapter 2). The vector takes multiple values for a single character class; all distinct feature vector are stored for a character. The feature vector in not unique for a character class; many character exhibit the same vector. The maximum number of characters associated with one feature vector observed is 10 and the character क has maximum number of distinct feature vectors at 18.

- *Moments: KS5-2*

Moments upto degree-2 are used for ranking the candidate characters. This is a direct application of moments as described in [1, 11, 33]. Each character is represented by 5 moment values. Mean and variance for these moment values for each character is computed from the values observed during the training phase (see section 2.7.2 of chapter 2).

- *Pixel-Density in 9-Zones: KS5-3*

The pixel density in 9-zones forms this feature vector. This feature is used only for upper and lower modifiers. (see section 2.7.4 of chapter 2).

3.1.6 Aspect Ratio: KS5-4

The aspect ratio divides the character set into 5 or 6 sets which have some overlap. For a known font, this feature is used for reducing the candidate set. (see section 2.7.3 of chapter 2).

3.1.7 Structural Prototypes for Characters: KS6

This KS consists of two features: structural representation and number-position of end points.

■ *Structural Representation of Characters: KS6-1*

The structural description schema has been briefly described in section 2.7.6 of chapter 2. The schematic is new which takes advantage of the script-specific features. Each character is represented by one or more descriptions. This feature is also used for ranking the candidate characters. (see section 2.7.3 of chapter 2).

■ *Number and Position of End Points: KS6-2*

Number and position of end points are also used for ranking the candidate characters. The character image is divided into 9(3x3) zones; the number of end points in each zone forms the feature vector. It is not a unique feature; number and position of end points is same for many characters. Due to noise and natural variation, number and position of end points also vary. All distinct feature vectors exhibited by a character during training phase are stored as alternative prototypes of the character.

3.1.8 Confidence Figure: KS7

The confidence figure associated with a candidate character is incremented after applying a classifying KS if the KS supports the candidate character (see section 2.8 of chapter 2). This knowledge source is transient.

3.1.9 Script Composition Rules: KS8

We decompose two dimensional Devanagari word into a linear sequence of recognition units. All the units after classification are composed back into syntactically correct Devanagari words. A subset of rules described in [71] are used for composition of words. These rules are embedded in the composition module. (see section 2.11 of chapter 2 and section 6.1 of chapter 6).

3.1.10 Word Envelop Information: KS9

The word envelop information extracted from the word image prior to character classification is done. The word envelop information is used for generating hypotheses for character boxes of the word (see chapter 6 for details).

3.1.11 Word Dictionary: KS10

A dictionary containing approximately 22,000 words is used for post-processing. The partitioning strategy is described in section 6.3.1 of chapter 6.

3.1.12 Character Confusion Matrix: KS11

IOCR tends to make certain substitution errors more frequently than others. Knowledge of such errors is used to select an alternative word from the dictionary whenever an exact match is not found (see section 2.9 of chapter 2 and section 6.5 of chapter 6)

Table 4 is a reference table for the name and number of various KSs.

KS Number	Knowledge Source
KS1	Threshold Line Height
KS2	Header Line
KS3-1	Threshold Core Character Height
KS3-2	Threshold Core Character Width
KS4-1	Coverage of the Core Strip
KS4-2	Position of the Vertical Bar
KS4-3	Number of Junction with the Header Line
KS4-4	Symbol Joining Patterns
KS5-1	Modified Zero-Crossings Vector
KS5-2	Moments
KS5-3	Pixel Density in 9-Zones
KS5-4	Aspect Ratio
KS6-1	Structural Representation
KS6-2	Number and Position of End Points
KS7	Confidence Figure
KS8	Script Composition Rules
KS9	Word Envelop Information
KS10	Word Dictionary
KS11	Character Confusion Matrix

Table 4: Reference Table for name and number of various Knowledge Sources.

The KSs are interfaced with the system through a set of tools. The result of invoking a KS is made visible to the entire set of KSs by posting the outcome on the solution blackboard. The solution blackboard is discussed next.

3.2 The solution blackboard

Solution blackboard is shown in figure 12. The binary image of document is at the lowest level. Word hypotheses are at the top level. The image segmented into characters and symbols (see chapter 4) are placed at the level above the text image. The characters and symbols of a word obtained by the preliminary segmentation module are posted on the blackboard. The segmented units are linearized. Initial hypotheses for each character/symbol are posted on the blackboard. Various transient statistics information is posted on the blackboard as they become available (not shown in the figure) which in turn trigger creation of corresponding transient knowledge sources. As the recognition process progresses, different knowledge sources are invoked which filter out some of the hypotheses and associate a confidence figure with the remaining hypotheses. These are composed into words which are verified. An example has been used to show the contents of the solution blackboard in figure 13.

3.3 Control Strategy and Process Interaction

Our document reading system segments words from a document and posts the image on the blackboard. The image of the word is further segmented and the processing starts. The initial program - *StartOff* is given in figure 14.

The middle level control program - *InvokeOthers* is activated by the *StartOff* program after symbol image boxes are posted on the blackboard. The middle level control program is shown in figure 15. It returns control to the initial control program after processing all the character images posted on the blackboard. It invokes an appropriate control process based on the type of the image box (core character, upper modifier, lower modifier). The output of the *CoreRecognizer* in terms of the confidence figure associated with each hypothesis is checked. If none of the candidate characters has an acceptable confidence figure and the image is conjectured to be a composite character image, further segmentation is done (see

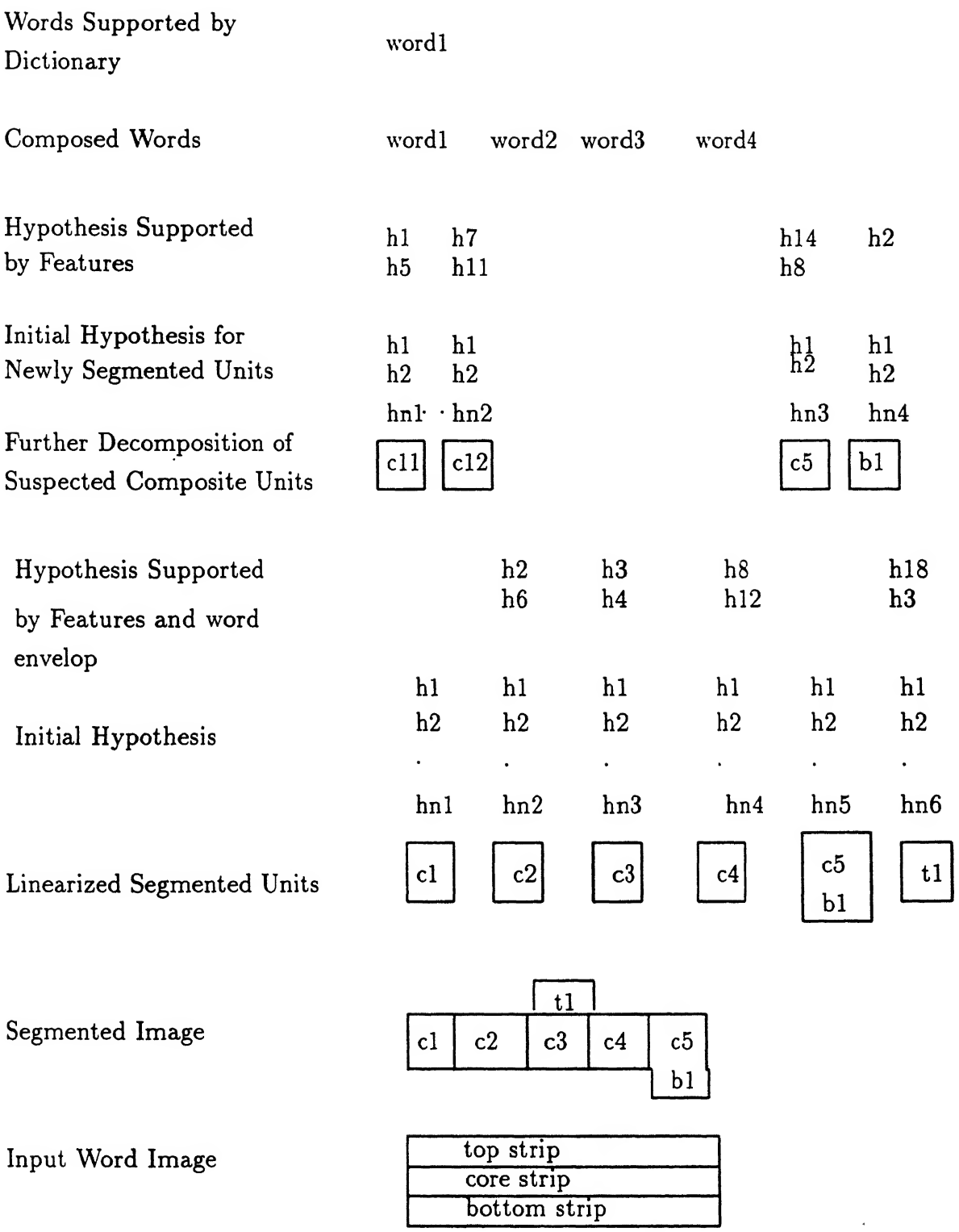


Figure 12: Solution Blackboard For Devanagari Text Analysis and Recognition System.

Words Supported by Dictionary			अध्येता			
Words Composed from top two hypotheses	अध्येता, अध्येला, अध्मेता, अध्मेला, अप्येता, अप्मेता अप्येला, अप्मेला, क्षध्येता, क्षध्येला, क्षध्मेता, क्षध्मेला क्षप्येता, क्षप्मेता, क्षप्येला					
Hypotheses Supported by Features in decreasing order of confidence	ध ढ म ट भ ध	य म प भ स अ				
Initial Hypotheses for Newly Segmented Units	all of core ch.	all of core ch.				
Further Decomposition of Suspected Composite Units	ध	य				
Hypotheses Supported by Features & word envelop in decreasing order of confidence	अ क्ष स म व य	*	त ल न व ब ठ	। - - - - -	ˋ ः ॰ ॰ - -
Initial Hypotheses	all of core ch.	all of core ch.	all of core ch.	all of core ch.	all of top sy	
Linearized Segmented Units	अ	ध्य	त	।	ˋ	
Segmented Image	अ	ध्य	ˋ	त	।	
Input Word Image			अध्येता			

Figure 13: An Example showing Contents of the Solution Blackboard For Devanagari Word Recognition, asterisk represents a rejected character.

```
process: StartOff

extract uniform text zone from the document page
for each uniform text zone
{
   tool: Line Identification
     segment uniform text zone into text lines: phaseI
        [ produce KS1 ]
     segment uniform text zone into text lines: phaseII
        [ use KS1 ]
   for each text line
     tool: Word Isolation
       segment text line into words
          [ use KS2 ]
       for each word
         tool: Symbol Extraction: PhaseI
         separate top strip from
              the remaining (core + lower)  strip
              [ USE KS2 ]
         segment top strip into units that are
              vertically separate from there neighbours
         segment the remaining strip into units that
              are vertically separate from there neighbours
        analyze height and width of units of
                          (core + lower) strip
           [ produce KS3-1, KS3-2 ]
    for each word
      post symbol image boxes on the blackboard
      post  initial candidate sets
extract word envelop information
          [ produce KS9]
generate character  Hypotheses
          [ use KS9, KS10 ]
post the hypotheses set on the blackboard
       transfer control to InvokeOthers program
 }
```

Figure 14: Initial Control Program.

chapter 4). The segmented image is appended on the blackboard. After processing every image constituting a word, the output is composed back into word(s). Words are then checked in the dictionary and corrected if necessary.

Recognition process is different for characters/symbols of each strip. The lower modifiers are recognized by *LowerRecognizer* and upper modifiers by *UpperRecognizer*. These processes are shown in figures 16 and 17 respectively.

The recognition process - *CoreRecognizer* for the core characters is invoked once the core character image is posted on the blackboard. The testing phase reveals the credibility of various classifying features. The features which are reliable and robust are used as filters. The following features for the core characters are robust:

- region of the core strip covered by the character in the core strip: KS4-1,
- presence and position of the vertical bar: KS4-2,
- junction with header line: KS4-3,
- modified horizontal zero crossing: KS5-1.

It is ensured that the true character is not missed by allowing other characters in each class.

The following features are used for ranking the candidate characters:

- moments: KS5-2, (for the known font),
- aspect ratio: KS5-4, (for the known font),
- structural representation: KS6-1,
- Number and position of vertex points: KS6-2.

```
process: InvokeOthers
for each symbol box of the word
{  if ( core character )
       Invoke CoreRecognizer
       if ( confidence of no hypothesis is above CONF_THRESH  or
                   hypotheses set is empty)
          [use KS7]
          tool: Symbol Isolation: Phase II
          if (width of the character > thresh_width)
             [ use KS3-2 ]
             IdentifyType
             if ( conjunct )
                SegmentConjunct
                  [ use KS4-4 ]
                post segmented units on the blackboard for processing
             else if ( shadow character )
                SegmentShadowPair
                post segmented units on the blackboard for processing
          tool: Symbol Isolation: Phase III
          if (height of the character > thresh_height)
                [ use KS3-1 ]
             SegmentLowerModifier
             post segmented units on the blackboard for processing
    else if ( upper symbol )
      UpperRecognizer
    else if ( lower symbol )
      LowerRecognizer
   tool: Character Pair Disambiguating Expert
   Disambiguate target character pairs
   tool : Word Composition
   ComposeWord
     [ use KS8 ]
   tool: Word Hypothesis Generation and Verification
   Verify and Correct Word
     [ use KS7, KS10, KS11 ]
```

Figure 15: Control Plan which invokes Control Plan for Lower Modifiers, Upper Modifiers or Core Character based on the type of the image box posted on the Blackboard.

```
process: UpperRecognizer

tool: Symbol Recognition: I
 if ( image of a top symbol has been posted
                                  on the blackboard )
 {
   compute modified horizontal zero crossing
                   feature vector
   eliminate some of the hypotheses from
                   initial hypothesis set based
                     on above feature vector
         [ use KS5-1 ]
   locate end points and their locations
   eliminate more hypotheses from the blackboard
                   based on above feature
         [ use KS6-2]
   compute 9box feature vector
   eliminate more hypotheses from the blackboard
                   based on above feature
         [ use KS5-3 ]
 }
```

Figure 16: Recognition Path for Upper Modifiers.

When an invoked KS supports a hypothesis, it increments the confidence figure of the hypothesis. The increment is indirectly proportional to the distance obtained by matching the feature of the unknown character with the stored prototype for the hypothesis. The *CoreRecognizer* process is outlined in figure 3.3.

The *Line Identification, Symbol extraction* and *Symbol Recognition* processes consist of more than one subprocesses. We have referred to these processes as *Phase I, Phase II* etc. Each of these phases have been elaborated upon in section 7.1 of chapter 7.

Character Pair Expert: Character व and ब are so similar that IOCR output

```
process: LowerRecognizer

tool:  Symbol Recognition: II
 if ( image of a lower symbol has been posted
                                  on the blackboard )
 {
   compute modified horizontal zero crossing
                   feature vector
   eliminate some of the hypotheses from
                   initial hypothesis set based
                     on above feature vector
       [ use KS5-1]
   compute 9box feature vector
   eliminate more hypotheses from the blackboard
                   based on above feature

       [ use KS5-3 ]
 }
```

Figure 17: Recognition Path for Lower Modifiers.

cannot be relied upon. Whenever, the output is either व or ब, a character pair expert is invoked to closely examine the image and give a decision in favour of one of the two. The middle region of the image, which has the discriminating stroke, is checked using horizontal zero crossings. In case of व, a sequence of the type $1_i 2_j 1_k$ is expected where $i, j, k > 0$. In case of ब, the sequence $1_i 2_j 3_k 2_l 1_m$ is expected where $i, j, k, l, m > 0$.

The character य is confused with characters म and प. This confusion is resolved by checking the curvature of the left most stroke. In case of य, the stroke must move in the south-west direction. Whereas , in case of म and प, the stroke moves south and north-east only.

The character भ gets confused with म and प. This confusion is resolved by looking at the top region of the image. The number of zero crossings for भ is 3 whereas for

```
process: CoreRecognizer
tool:  Symbol Recognition: III
 if ( image of a core character has been posted on the blackboard )
   compute PlaceInCore feature
   eliminate some of the hypotheses from initial
         hypotheses set based on above feature vector
       [ use KS4-1 ]
   compute VertiBar feature
   eliminate some more hypotheses from the solution
          blackboard based on above feature vector
       [ use KS4-2 ]
   compute number of junctions with header line
   eliminate some more hypotheses from the solution
        blackboard based on above feature vector
       [ use KS4-3 ]
   compute modified horizontal zero crossing
                  feature vector
   eliminate some of the hypotheses from the solution
         blackboard based on above feature vector
       [ use KS5-1 ]
   itersect pruned set and hypotheses set
if ( the intersection set is null )
return control to the calling module
   initialize confidence figure to CONF_INIT
       [ produce KS7 ]
   compute end points and their location
   increase the confidence figure of  the hypothesis on
        blackboard supported by above feature vector
       [ use KS6-2 ] [ modify KS7 ]
   obtain  structural representation of the image
   increase the confidence figure of  the hypothesis
                 on blackboard according to the
                 distance from the prototype
       [ use KS6-1 ] [ modify KS7 ]
```

Figure 18: Recognition Path for Core Characters.

म and प, is it two. The म and प pair is resolved by looking at the zero crossings in the middle half of the image. The character pair expert is designed for each closely resembling character pair which classification process is unable to resolve all the time.

One of the design consideration for control strategy has been the nature of the problem itself. The problem does not change with time due to external events as in case of online document reading. Therefore, a dynamic control structure is not required.

The architecture facilitates experimentation with new features. A library of features can be built independently with access tools. The effect of each feature on the performance can be studied. It is possible to design an adaptive control strategy which learns from the environment. However, this has not been attempted here.

The use of a hierarchical solution blackboard provides a clean interaction mechanism among various modules. All control strategies mentioned above have been integrated to form the control mechanism of the document reading system.

The KSs corresponding to prototypes for various classifying features may be required by more than one processing path. The extracted features from the unknown character image are made available globally to all the processes.

For a multifont document, there are two possible approaches for classification process:

- Identify the font and use corresponding font prototypes
- Use font-independent features for classification

Identification of the font will enable the use of font specific features, such as moments, aspect ratio etc..

To be able to identify a font, we must know the discriminating properties of a font. At the top level, the following parameters are specified for a font:

Spacing
- Fixed spacing
- Proportional spacing

Pitch Number of characters in an inch, it only applies to fixed space fonts

Height A PCL point is 1/72inch. The body of the type is slightly greater than the distance from the bottom of a descender to the top of an unaccented capital letter. For example, Hg will give the height for this font

Style
- Upright
- Italic

Stroke Weight Thickness of the strokes that compose characters , Pen Style/Width This is either bold, medium or normal.

Width Type Specifies the proportionate width of characters in font

- Condensed
- Semi-Condensed
- Normal
- Semi-Expanded
- Expanded

Two different fonts may have all the above specified parameters same. Even if the parameters are different, it may not be easy to perceive the difference for two fonts of the same point size. Fonts differ from each other in more subtle ways.

MetaFont [42] is a system that is used to design a font. There is a concept of *pen* and *pen styles*. Sizes and shapes of pen nibs are varied and the outlines of each stroke are precisely controlled. Thickness, angle and intensity of a pen can be changed.

A curve is defined by two points through which the curve passes and two optional intermediate points. A curve may be changed by breaking it into more segments and by defining points for each segment. It has been said in [42] that mathematical principles fail to produce good letters. To quote:

> *Albrecht Durer and other Renaissance men attempted to establish mathematical principles of type design, but the letters they came up with were not especially beautiful. Their method failed because they restricted themselves to "ruler and compass" constructions, which cannot adequately express the nuances of good calligraphy.*

The implication is that font identification is not simple curve fitting. It is not easy to identify a font based on its characteristics. It becomes even more difficult when the characters are not known. Further, there are enormous number of fonts that exist and new fonts are also being introduced. Therefore, we selected a set of commonly used fonts to build the prototype library and made an effort to identify the font of the document assuming it to be one of the library fonts. The non-composite characters yielded by preliminary segmentation process are used for identification of the font. Vertical bar property and horizontal zero crossing vector are used as filters to reduce the set of candidate characters for each unknown character of a word. Now, font sensitive feature: moments 2 are used to get cardinal distance of the unknown character from the prototypes of each choice for each font library. The distance is calculated for many words. The font that gives minimum distance for majority of the characters is the font of the document. In case of ambiguity, more words are used for distance calculation. However, this experimentation was done for Roman script.

For Devanagari script, not much work has been done for standardizing the fonts which further discourages use of font dependent features. Therefore, a system has been developed that uses only font specific features when the font is known and the corresponding prototype library is available. Otherwise, the font-independent

features are used which include vertical bar property, modified horizontal crossings vector, structural representation and position of vertex points.

Chapter 4

Extraction of Units for Recognition from the Document Image

A document page may contain images, graphs, tables etc. in addition to the text. Extraction of text-zones from the document has been extensively studied [6, 10, 13, 96] and still continues to be an active research area. However, in this thesis, we assume the presence of a preprocessing stage that extracts uniform text zones from the document image. Our system segments each uniform text zone into text lines and text lines into words. Words are further segmented into characters and symbols. The characters and symbols may not be valid Devanagari symbols when viewed in isolation. We refer to characters and symbols as 'units for recognition' or 'recognition unit'[1]. In this chapter, each segmentation phase is described along with algorithms and examples. At each stage, transient knowledge of the text being processed and the structural properties of the script have been used.

[1]When it is clear from the context, only 'unit' has also been used.

4.1 Line Segmentation

Devanagari script is written from left to right, top to bottom. A text line is separated from the previous and following text lines by white space. There are situations when one or more top modifiers of a text line overlap with lower modifiers of the previous line. As a result, white space no longer separates a text line from its neighbours.

We have employed a two-pass mechanism to segment a uniform text zone into constituent text lines. In the first pass, lines are separated assuming that no line fusion or break exists. This segmentation is based on horizontal histograms of the document. A horizontal histogram of the uniform text zone is made. A zero value in the histogram corresponds to a horizontal gap. The horizontal gaps are assumed to be the line boundaries. Heights of all text lines obtained in first pass are divided into three bins. The maximum height of a segmented line is referred to as *MaxLineHt.* The first bin contains those lines whose height is 80% or more of the *MaxLineHt.* The second bin contains the text lines whose height is less than 80% of *MaxLineHt* and more than 64% of *MaxLineHt.* The remaining text lines go in the third bin. The number of text lines in each bin is counted. The bin which contains maximum text lines becomes the representative bin for properly segmented text lines. The average height of a text line in this bin is used as *threshold height.* A text line whose height is more than the *threshold height* is suspected to be the fusion of more than one text line. The suspected fused lines are further segmented in the second pass.

Second pass makes an attempt to break a suspected fused line into constituent lines using *threshold height.* The image is scanned from top to *threshold* height and the header line is located which is the most dominating horizontal line. In the region near *threshold height*, the minimum pixel strength position is used for breaking the fused lines. Algorithm is presented in figure 19.

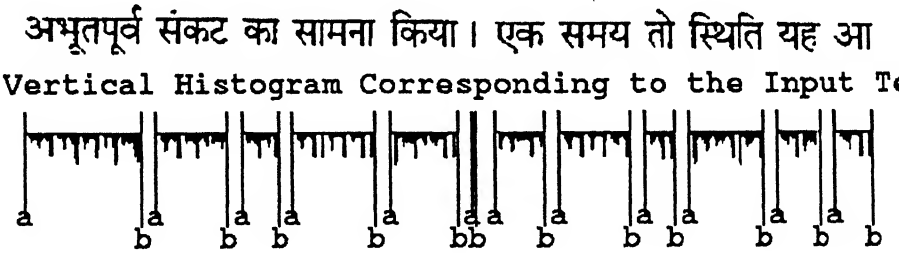
```
Algorithm:  Segmentation of uniform text zone into text lines
Input     :  text zone image
          :  numPixLine /* number of pixel lines in the text image */
Output    :  thHeight
          :  pos[MaxTxtLine][2]
        /* begin and end pixel line position of each text line */

  make horizontal histogram of the image: HHisto
  ln = 0
  while ( ln < numPixLine )
     txtLine = 0
     skip initial blank pixel lines
     pos[txtLine][0] = ln
     while ( ln < numPixLine  and hHisto[ln] > 0 )
       increment ln
     pos[txtLine][1] = ln
     increment txtLine
  divide line heights into bins  and set thHeight
  for each text line
    locate position of max value of hHisto in the top half,
    call it maxPos
    search for a row where hHisto is 0
          from maxPos to  thHeight - ( maxPos - top )
       if ( found )
          set pos[txtLine][1]
       else
          locate pos of the row with minimum no. of pixels
                to break the line in the above region
    start processing for next text line
```

Figure 19: Algorithm for segmentation of uniform text zone into text lines

4.2 Segmentation of a text line into Words

In Devanagari script all characters and symbols of a word are joined together by a header line. As a result, word boundaries are rarely ambiguous. However, the header line has a gap if any of following characters are present in the word: थ, ध, भ, क्ष. Some example words are सुरक्षा, पृथ्वी, आभार, धीरे. The gap in header line creates no problem for word boundary identification process as the gap in header line does not create a vertical gap in the word. A vertical histogram of a text line is made and every gap of two or more pixels in the histogram is taken to be the word delimiter. Segmentation of a text line into words is almost self-explanatory. Figure 20 shows a text line and its segmentation into words.

(a) Input Text Line:



(b) Vertical Histogram Corresponding to the Input Text Line:

Figure 20: Word boundary identification in a text line with the help of vertical histogram; (a) shows the image of the text line; (b) shows the vertical histogram of the text line image; line marked 'a' shows the left boundary of a word and line marked 'b' shows the right boundary of the word.

4.3 Segmentation of a Word into Symbols and Characters

It is easy to identify the word boundaries due to the header line of a word. However, the header line joins the characters of a word together which makes the segmentation

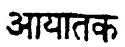
of a word into its constituent characters slightly involved. The region above header line contains upper modifier symbols. The region below the header line contains core characters and lower modifier symbols. Before segmentation can progress any further, header line must be identified. Header line is easily identified as it is the most dominating horizontal line in a word. After the header line is removed, vertical gap separates the top modifiers from their neighbours. The characters below header line are also separate from their neighbours by vertical gap. But a character obtained by this simple segmentation may have a lower modifier or may be a conjunct. This segmentation is termed as 'preliminary segmentation'. The conjuncts and composite characters are further segmented by a 'composite character segmentation module'. The composite character segmentation process uses structural properties of the script to segment the conjuncts and composite characters into constituent characters and symbols. A character yielded by the preliminary segmentation process is suspected to be a composite character based on the height and width of the character with respect to other characters. If the recognition process fails to classify a suspected composite character into known classes, further segmentation of the character is attempted. This is a two pass algorithm for segmentation and decomposition of Devanagari composite characters/symbols into its constituent symbols. The algorithm extensively uses structural property of the script. In the first pass words are segmented into easily separable characters/composite characters. Statistical information about the height and width of each separated box is used to hypothesize character boxes to be composite. In the second pass, the hypothesized composite characters are further segmented. The algorithm is designed to segment a pair of touching characters. In the following sections, we discuss preliminary and composite character segmentation of words.

4.3.1 Preliminary Segmentation of Words

All the symbols and characters of a word are joined together by the shirorekha. Therefore, before attempting segmentation, it is necessary to 'remove' header line. If header line is removed as a single stroke from the word, two problem surface. First,

there are some character pairs that differ in the header line region only. The header line is broken for one while it is complete for another. Two such examples are म भ and घ ध. After removing the header line , it becomes difficult to distinguish one from the other. Second, a small tilt in the orientation of a word, if present, may widen the header line. If header line is removed completely, some of the characters get chopped from the top. On the other hand, if only significant part of the header line is removed, the residual of header line becomes noise for the characters. Figure 21 shows image of words, image after removing entire header line and image after removing only significant part of the shirorekha.

(a) Word Image



(b) Word Image after removing the significant part of the shirorekha

(c) Word Image after removing the entire shirorekha

Figure 21: Removal of Shirorekha form a Word and Associated Problems.

To handle both these problems, header line is located by means of horizontal histogram of a word. But header line is removed from each character individually. To get the individual characters, few pixel region above and below header line is ignored while looking for vertical gap. The image above the header line and below the header line are handled separately. For each part of the image, a vertical histogram is made. In the vertical histogram, a column which contains no black pixel corresponds to a vertical gap between two neighbours. An algorithm based on the above ideas is given in figure 22.

Figure 23 shows a text line and its preliminary segmentation. Figure 23(a) shows a sample text line. The characters obtained after preliminary segmentation are shown in figure 23(b).

This algorithm has gone through extensive testing on printed documents and has

been found to be extremely reliable. We have not observed any case when the word boundary identification was wrong.

4.3.2 Transient Statistics about Height and Width of Core Characters

A text page written in Hindi which uses Devanagari script has less than 5% composite characters. Statistics about width and height of characters obtained by preliminary segmentation is used to mark the characters boxes which are likely to be composite character boxes. All characters are divided into three different bins based on their height. The maximum height of a segmented character box is referred to as *MaxCharHt.* The first bin contains those character boxes whose height is 80% or more of the *MaxCharHt.* The second bin contains the character boxes whose height is less than 80% of *MaxCharHt* and more than 64% of *MaxCharHt.* The remaining character boxes go in the third bin. The number of character boxes in each bin is counted. The bin which contains maximum character boxes becomes the representative bin for properly segmented character boxes. The average height of a character box in this bin is placed in *threshold character height.* All characters which have height more than *threshold character height* are marked as characters which may have the lower modifiers. In the same manner, *threshold character width* is also computed. All characters which are wider than *threshold character width* are suspected to be either shadow characters or conjuncts.

The height and width of the characters obtained after preliminary segmentation is shown in the figure 23(c, d) followed by *threshold height* and *threshold width.* Based on these threshold values, the suspected composite character boxes are marked which are shown in figure 23(e).

```
Algorithm:  Preliminary segmentation of a word(contd.)}

   lPos = headerLinePos - imhHt * 0.10
   Make vertical histogram VHisto  of image from top of word to lPos
   i = 0
   scan every column of VHisto from left to right
   if ( column contains no black pixels)
     move to the next column
   else
     box[charCount][0] =  column position
     advance column position till a column containing
          no black pixels or word boundary is reached
     set box[charCount][1]  to current column position
     make horizontal histogram CHHisto  of the image enclosed by top
         and bottom boundaries of the word and newly located
         left and right;
     scan CHHisto from top to bottom and locate row containing
          max no. of pixels in chHHisto
          call it maxRow and the set maxVal to no. of pixels
     check the previous rows  starting from maxRow  till a row
         containing 70% of maxVal is reached
      set box[charCount][4]  to current row position
      set box[charCount][3]  to the top of the word
      increment charCount
```

Figure 22: Algorithm for preliminary segmentation of a word.

(a) Input Text Line:

यह हमारा भारत राष्ट्र है ।

(b) Characters and Symbols obtained after Preliminary Segmentation

य ह ह म ा र ा भ ा र त र ा ष्ट्र ह ै ।

(c) Height of the Core Characters

31 43 42 31 33 31 34 32 31 32 32 32 27 45 42 32

(d) Width of the Core Characters

30 25 25 31 6 22 6 28 5 22 27 22 6 40 24 6

Threshold Height for the Core Characters:33

Threshold Width for the Core Characters:38

(e) Characters Suspected to be Composite Characters

ह ह ष्ट्र ह

Figure 23: Preliminary segmentation of a sample text line; (a) Input text line; (b) Characters and Symbols obtained after preliminary segmentation; (c) Height of the core characters in pixels; (d) Width of the core characters in pixels; (e) Characters suspected to be Composite Characters.

been found to be extremely reliable. We have not observed any case when the word boundary identification was wrong.

4.3.2 Transient Statistics about Height and Width of Core Characters

A text page written in Hindi which uses Devanagari script has less than 5% composite characters. Statistics about width and height of characters obtained by preliminary segmentation is used to mark the characters boxes which are likely to be composite character boxes. All characters are divided into three different bins based on their width. The maximum height of a segmented character box is referred to as *MaxCharHt.* The first bin contains those character boxes whose height is 80% or more of the *MaxCharHt.* The second bin contains the character boxes whose height is less than 80% of *MaxCharHt* and more than 64% of *MaxCharHt.* The remaining character boxes go in the third bin. The number of character boxes in each bin is counted. The bin which contains maximum character boxes becomes the representative bin for properly segmented character boxes. The average height of a character box in this bin is placed in *threshold character height.* All characters which have height more than *threshold character height* are marked as characters which may have the lower modifiers. In the same manner, *threshold character width* is also computed. All characters which are wider than *threshold character width* are suspected to be either shadow characters or conjuncts.

The height and width of the characters obtained after preliminary segmentation is shown in the figure 23(c, d) followed by *threshold height* and *threshold width.* Based on these threshold values, the suspected composite character boxes are marked which are shown in figure 23(e).

4.3.3 Type Identification of Composite Characters

A composite character is unlikely to get classified to known classes by the recognition process. But the possibility of a mapping of a composite character to another known class cannot be ruled out. Therefore, characters which are marked as likely composite characters by preliminary segmentation process are further segmented if:

1. The confidence figure associated with classification is below a preset threshold,
2. Classification to the known classes fails or
3. Hypotheses generated by using envelop information and candidate set generated based on gross features of the character box image have no element in common.

A wide character could be a conjunct or a shadow character, both of which require different treatment for further segmentation due to the difference in their formation. Two characters in shadow do not touch each other. Whereas the constituent characters of a conjunct touch each other. This knowledge is used to distinguish a conjunct from a shadow character. Outermost boundary of the image is traversed starting from left most black pixel. The traversal covers the entire width and height of the image if the constituent parts are touching. If the constituent parts are not touching, the traversal covers the first component and as a by-product, gives the coordinates of the first constituent character.

For outer boundary traversal [16], first step is taken in the north_west direction from the initial point, if possible. If northwest neighbour is not present or the image boundaries are reached, next neighbour is tried. The algorithm outlined in figure 24 gives the next direction of traversal for an outer boundary traversal. Eight possible directions are shown in figure 25. In this figure, a number is used for each direction instead of name.

```
dir = 3;
do
{
  dir = ( dir + 4 ) mod 8;
  if ( neighbor in dir is not present or
       it is outside the image boundary )
     dir = ( dir + 1 ) mod 8;
}until initial point is not reached
```

Figure 24: An algorithm for deciding direction of next step for outer boundary traversal.

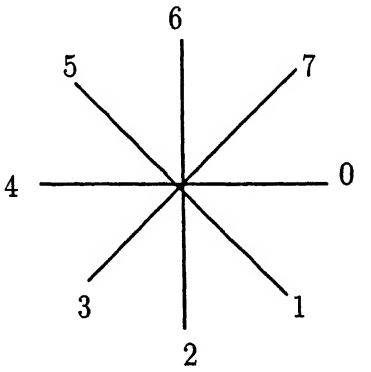


Figure 25: 8-neighbours and the direction number.

4.4 Segmentation of Shadow Character

Two adjacent characters in shadow cannot be separated by a single vertical line. Let us refer to the left and right boundaries of the enclosing box of the shadow characters by *left* and *right*. The shadow character segmentation algorithm locates two vertical lines corresponding to the right boundary (referred to as *right1*) of the first constituent character and left boundary (referred to as *left2*) of the second constituent character. The box enclosed by *left* and *right1* will contain part of the other character. Similarly, the box enclosed by *left2* and *right* will contain part of the other character. The unwanted part of the image of the other character is the

overlapping region. The overlapping part is cleaned to obtain the character images.

An outer boundary traversal done for identifying the type of the composite character yields the coordinates of the first character box. The right most point of the first character image is the *right1*. To obtain *left2*, another traversal is initiated starting with a point which is outside the first character box. During this traversal, the extreme left, right, top and bottom points which are visited in the territory of first box form the overlap box for the first character. Another traversal of first character with additional knowledge of the coordinates of the second character is done to locate the overlap region for the second character box. Some of the characters segmented by an algorithm using above idea are shown in figure 26.

The shadow character separation introduces no noise and therefore the performance of the system is not affected by shadow character separation. However, if shadow characters remain unsegmented and get substituted by some other character, then the performance is affected. During testing phase, it was observed that the non-touching characters remain unsegmented after preliminary segmentation primarily due to the presence of lower modifiers. Whenever the lower modifier expands over two adjacent character boxes, the vertical separation of the character boxes cannot be done until the lower modifier is separated. Such character boxes are separated right after the segmentation of lower modifiers. Among the shadow character pairs, the character pair एक is the most frequent shadow character pair whose constituent characters are ए and क. It is obvious that finding the continuity of white pixels would separate ए in two segments. Whereas, our algorithm does not have this problem because we assume that the overlap between the constituent character boxes is partial. Therefore, the search for the second constituent character is done outside the first character box.

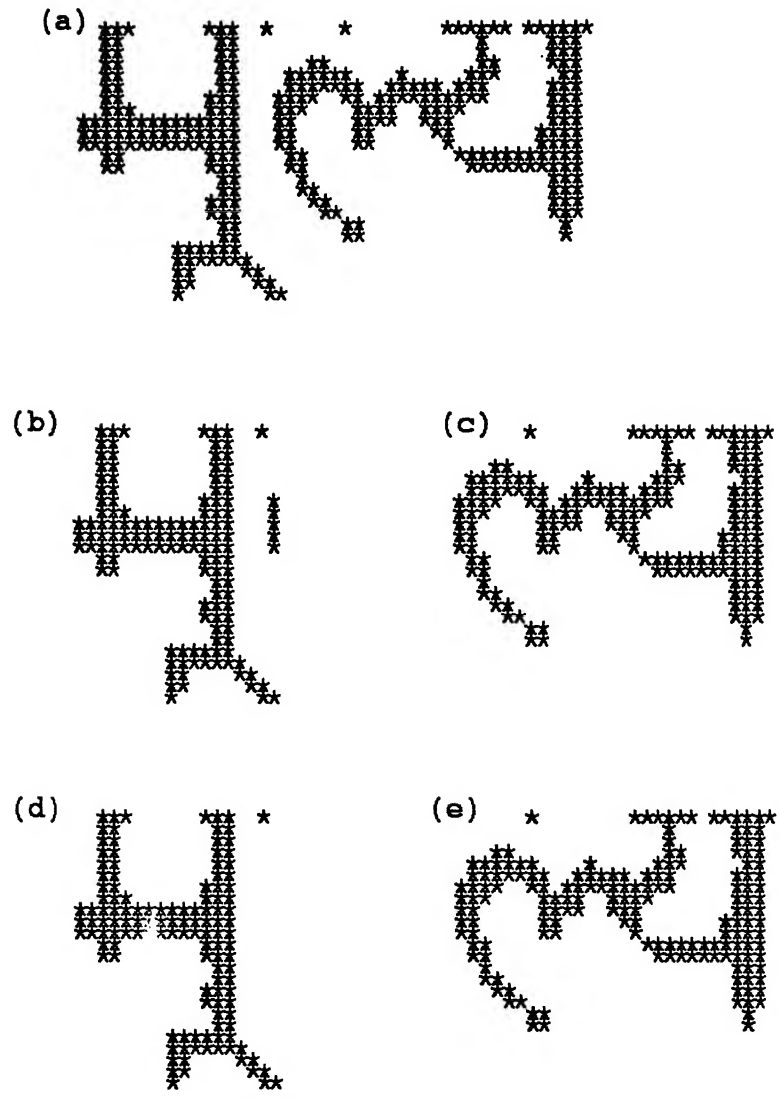


Figure 26: Two Characters in Shadow and their Segmentation; (a) The image of Characters in Shadow; (b) The box enclosing the first character contains part of the second character; (c) The box enclosing the second character contains no part of the first character; (d) and (e) show the constituent characters after cleaning the overlapping region.

4.5 Segmentation of Lower Modifiers

Devanagari script has characters of varying height. Character ह has almost same height as ह्‌ which is character द with the lower modifier ्. Therefore, a character which is suspected to have a lower modifiers may not actually contain a lower modifier. When a lower modifier is placed below a core character, one of the three possible joining patterns is formed. These three joining patterns are:

i. **Weak Joining Point:** A lower modifier below a middle or end bar character is weakly joined with the core character. Some of the no-bar character that have a small bar in the lower region also form a weak joining point with the lower modifier. Some examples are हु, कू, घु.

ii. **Thick Joining Point:** There are some characters that touch the lower end of the core strip at more than one point. A lower modifier placed below such character forms a thick join with the core character, such as खु, सू.

iii: **Gap between character and modifier:** Sometimes a modifier does not touch the core character at all. The diacritical mark called *nukta* (represented as a *dot*(.) below the carrying character) and the lower modifier symbol *halant* usually do not touch the core character. Some example characters with *nukta* and *halant* are: ड़ ढ़ क्.

The region for lower modifiers is below the core strip. Height of the core strip is approximately *thHeight* (see section 4.3.2). If the lower modifier is separate from the core character, no additional processing is required. Otherwise, weakest point near the end of core strip is located. The pixels below the weakest point are checked whether they have sufficient height and width to qualify for a lower modifier symbol. Sufficient height of the lower modifier is 1/5th of the core character height. In case, enough pixels are present, an attempt is made to identify the joining pattern. If a thick joining point is found, a break point near *threshold height* is generated with

the help of profile information of the joining region. An example of the character before and after separation of lower modifiers are shown in figure 27.

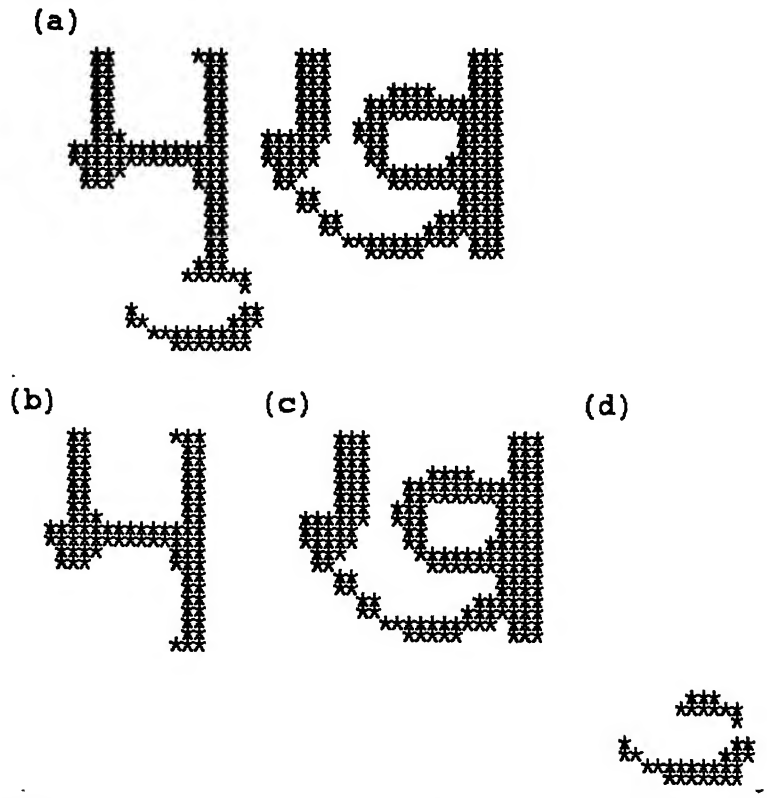


Figure 27: Lower Modifier Segmentation; (a) The image of a character containing a lower modifier; The following character is in shadow of the lower modifier; (b) First character after separation of lower modifiers; (c) The following character is also extracted; (d) The lower modifier.

The character ह्न is a long character and it has a high frequency of occurrence in the text. The image below the *threshold height* resembles the lower modifiers *halant* very closely. Our lower modifier segmentation algorithm removes lower part of ह्न and puts it in a lower modifier box. If the unsegmented ह्न is classified first then the segmentation will not be attempted and the problem caused by unnecessary segmentation of ह्न will not surface. If the lower modifier separation is done before the classification is done and the unsegmented ह्न is also retained, then the classification process will classify the unsegmented ह्न and the segmented images will not be used. However, if only the segmented version is retained, the chopped character is essentially a new character and it is treated like a new character. This lower modifier does not make a syntactically valid character with character ह्न. Since the

trainer and recognizer both use the same algorithm, character ह्‌ is always chopped. The chopped character ह्‌ is added to the set of known characters. The composition grammar is augmented to ignore (see chapter 6) the lower modifier ् when the carrying character is ह्‌.

In our implementation, we segment a character box (say CBox1) for lower modifiers without first attempting the classification. We retain the unsegmented version, CBox1, and store the segmented image boxes (in CBox2, CBox3, etc.) as an alternate of CBox1. We first try to classify CBox1 and if the classification succeeds, the image boxes CBox2, Cbox3 etc. are ignored. If the classification of CBox1 fails, we try to classify the image boxes CBox2, CBox3 etc..

4.6 Segmentation of Touching Character

We refer to the conjunct image by image(conj_left, conj_right, conj_top, conj_bottom) where conj_left, conj_right, conj_top and conj_bottom are the left, right, top and bottom coordinates of the minimum sized upright bounding box of the image. Before discussing the algorithms, we need to define the following:

1. **Vertical Projection:** The pixel projection has been defined in [48] as $\{PXP(k), k = 1, 2, \ldots, W\}$. It consists of total number of black pixels in each vertical column and W is the width of the image. We refer to this projection as vertical projection. The vertical projection of the selected part of the image is obtained by specifying the co-ordinates of the selected part of the image.

2. **Collapsed Horizontal Projection:** The collapsed horizontal projection is defined as $\{HP(k), k = 0, 1, \ldots, H\}$ where HP(k) records the presence of black pixels in each horizontal row and H is the height of the image. The collapsed horizontal projection of selected part of the image is obtained by specifying the co-ordinates of the selected part of the image. Therefore, collapsed horizontal projection $\{HP(k)\}$ of image(conj_left, conj_left+(conj_right - conj_left)/3,

conj_top, conj_bottom) is the collapsed horizontal projection of left 1/3rd part of the image.

3. **Continuity of the Collapsed Projection:** We scan collapsed horizontal projection$HP(k)$ in the sequence $HP(0), HP(1), \ldots, HP(W)$ and record the position of first row containing a black pixel; let this position be i. We continue the scan and record the position of first row containing no black pixels; let this position be j. We continue the scan and the collapsed horizontal projection $\{HP(k), k = 0, 1, \ldots, W\}$ has no discontinuity if all the rows below j contain no black pixels or j is the boundary row. The difference $j - i + 1$ is referred to as height of the projection.

4. **Vertical Bar and its Location:** Devanagari characters can be divided into three groups based on the presence and position of a vertical bar, namely: no bar characters, end bar characters and middle bar characters. Some of the characters belonging to each of these classes are shown in figure 28. The

end bar characters

अ ख घ च ज झ ञ त थ ध न प ब भ म य ल व ष स

middle bar characters

ऋ क फ

non-bar characters

इ उ ऊ ए छ ट ठ द ड ढ र ह

Figure 28: Three classes of core characters based on the vertical bar feature

position of the vertical bar is the left most column where number of black pixels is 80% or more of the character height. This is determined by scanning the vertical projection of the image from left to right. The first column where target number of pixels are present becomes the position of the vertical bar.

5. **Pen Width** In Devanagari characters, pen width is same as the thickness of the header line. Thickness of the header line is obtained during the preliminary segmentation process from the horizontal histograms of the words (see figure 21).

The conjunct segmentation algorithm process takes the image of the conjunct and the co-ordinates of the enclosing box. The position of the vertical bar and pen width are also inputs to the algorithm. The algorithm is outlined in figure 29.

We calculate height and width of the conjunct image referred to as *conj_height* and *conj_width* respectively. We also calculate left one third and mid columns of the conjunct image referred to as *left_onethird* and *conj_mid* respectively. Next, we establish the class of the first constituent character. Derived half form of the consonants are divided into two classes based on their heights:

Height of the derived half form is less than 80% ofthe height of the base form: Characters of this class are: ट ठ ड ढ ण थ द ध न ब भ म ल ळ ष ट . This class is referred to as class H_1.

Height of the derived half form remains the same as of the base form: Some such examples are: क रु ह्म फ र . This class of characters is referred to as class H_2.

The class of the first constituent character is determined by making collapsed horizontal projection of left 1/3rd of the conjunct image which is stored in *HPl1by3*. If the height of the *HPl1by3* is more than .8 * height of the conjunct image, the first constituent character of the conjunct belongs to class H_2. Otherwise, it belongs to class H_1.

For each class, we examine the image further to get the right boundary of the first constituent character. For class H_1 characters, we again check the height of *HPl1by3*. If the height is less than 1/3rd of conjunct height, we include additional columns

in steps of one in image under consideration and modify *HPl1by3*. The inclusion of additional columns stops when the required number of rows with black pixels is obtained or the middle column of the conjunct is reached. If the middle column of the conjunct has been included in *HPl1by3*, the middle column becomes the right boundary column of the first constituent character of the conjunct. Otherwise, from the last column included in *HPl1by3* to middle of the conjunct image, we examine each column with respect to the next one. This is done by making vertical projection of the required columns. If the pixels strength in the next column is more than the pixel strength in the column under examination, we make the present column the right boundary column of the first constituent character.

In case of class H_2 characters, we look at the left one third of the image and locate the right most column which contains 50% or more black pixels of conjunct height. Starting from this column to the middle column of the conjunct image, we examine each column with respect to the next one. We make vertical projection of the required columns. If the pixels strength in the next column is more than the pixel strength in the column under examination, we make the present column the right boundary column of the first constituent character.

For extracting the second constituent character of the conjunct, the connectedness property of the characters of the script is used. Let us refer to the box enclosing the character image as *ch_right, ch_left, ch_top, ch_bottom.* The character image is a single connected entity. In case, the character has a vertical bar at some position *bar_pos*, we shift *ch_right* to its immediate left; i.e. *bar_pos1 - 1.* Refer to this new value of right boundary of the character as *temp_right.* This amounts to ignoring the vertical bar and the image to its right. The character image surrounded by *ch_left, temp_right, ch_top, ch_bottom* remains connected. However, the image surrounded by *ch_left* + δ, *temp_right, ch_top, ch_bottom* is no more connected where δ is a positive integer greater than or equal to *penwidth.*

Therefore, the rightmost column to the left of *temp_right* in case of bar characters or *ch_right*, where the image component becomes connected, is the left boundary of

the second constituent character of the conjunct.

Returning back to our algorithm of figure 29, the search for the left boundary of the second constituent character starts from conj_right if no bar is present and immediately to the left of the bar if it is present. This point is referred to as *temp_conj_right.* We make an initial guess for the left column of the second constituent character. We refer to this column to as *temp_left2* and set it to the *temp_conj_right - 2 * penwidth.* We now make a collapsed horizontal projection of the image enclosed by *temp_conj_right, temp_left2, conj_top, conj_bottom.* This is referred to as *HPr.* If *HPr* has no discontinuity and the height of *HPr* is more than 1/3rd of *conj_height*, *temp_left2* becomes the left boundary of the second constituent character (referred to as *left2*). Otherwise, we move *temp_left2* to further left in steps of one column and modify *HPr.* We stop moving *temp_left2* further if *HPr* has no discontinuity and required number of rows are present. The present value of *temp_left2* becomes the left boundary of the second constituent character. However, if *temp_left2* has reached *left_onethird* and still a break column has not been located, the search is abandoned and no segmentation point is suggested.

If *left2* is less than *right1*, both the segmentation points are ignored and no segmentation is done. If *right1* and *left2* are same or the difference *(right1 - left2)* is same or less than double of pen width, the segmentation is accepted. However, the *right1* is moved to the left by the pen width. The projection *HPr* becomes connected as soon as every row has one black pixel and does not take *penwidth* into consideration. The correction in left2 takes care of this error.

4.7 Identification and Removal of *Rakar* Modifier Symbol

Seventeen out of thirty three consonant of the script have a *Rakar* form (see appendix A). The characters in rakar form along with their base form are shown below:

```
Algorithm: Segment Conjunct
Input:
  image: conjunct image
  int conj_left    /* left boundary of the conjunct image */
  int conj_right   /* right boundary of the conjunct image */
  int conj_top     /* top boundary of the conjunct image */
  int conj_bottom  /* bottom boundary of the conjunct image */
  /* conj_right > conj_left and conj_bottom > conj_top    */
  bar_pos:   position of the vertical bar in the second half
             of the conjunct image; If vertical bar is  not
             present, it is set to -1
  penwidth: thickness of the header line
Output:
  right1: right boundary of the first constituent \ch
  left2:  left boundary of the second constituent \ch

Procedure:
  conj_width = conj_right - conj_left + 1
  conj_height = conj_bottom - conj_top + 1
  left_onethird = conj_left + conj_width/3
  conj_mid = conj_left + conj_width/2

  make collapsed horizontal projection  of the image
    surrounded by (conj_left, left_onethird, conj_top, conj_bottom)
    store it in HPl1by3
  If ( height of HPl1by3 >= .8 * conj_height )
     class = h2
  else
     class = h1
```

Figure 29: Algorithm for locating the right boundary of the first constituent character and left boundary of the second constituent character of a conjunct (contd).

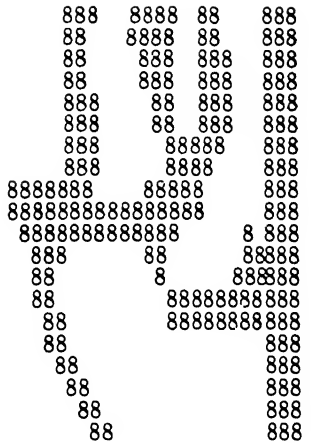
```
if ( class = h1 )
   add_column = 1
   while ( left_onethird + add_column < conj_mid )
   { If ( height of HPl1by3 <= .3 * conj_height )
        increment add_column by 1
        modify HPl1by3 to correspond to collapsed horizontal projection
           of the image enclosed by
            conj_left, left_onethird+add_column, conj_top, conj_bottom
     else break /* exit from while loop */
   }
   this_col = left_onethird + add_column
   right1 = this_col
   while ( this_col < conj_mid )
   { if ( number of black pixels in this_col <=
               number of black pixels in this_col+1 )
         increment this_col by 1
     else
        right1 = this_col; break /* exit while loop */
   }
else /* class h2 */
   this_col = conj_left + 1
   ht_col = left_onethird
   while ( this_col < left_onethird )
   { if ( number of black pixels in this_column >= .5 * conj_height )
        ht_col = this_col
        increment this_col by 1
   }
   this_col = ht_col + 1
   right1 = this_col
   while ( this_col < conj_mid )
   {  if ( num_black_pixels in this_col <= num_black_pixels in this_col+1 )
        increment this_col by 1
      else
        right1 = this_col; break /* exit from while loop */
    }
```

Figure 29: Algorithm for locating the right boundary of the first constituent character and left boundary of the second constituent character of a conjunct (contd).

```
if ( bar_pos > -1 )
  temp_conj_right = bar_pos - 1
else
  temp_conj_right = conj_right
temp_left2 = temp_conj_right - penwidth * 2
make collapsed horizontal projection HPr of image enclosed by
   (temp_left2, temp_conj_right, conj_top, conj_bottom)
while ( temp_left2 >= left_onethird )
{
   if ( HPr has no discontinuity and
             height of HPr > 1/3 of conj_height)
  {  left2 = temp_left2
     break /* exit from while loop */
  }
  temp_left2 = temp_left2 - 1
}

if ( left2 < right1 )
{ left2 = -1
  right1 = -1
  /* no segmentation possible */
  exit /* from the segmentation procedure */
}
if ( left2 - right1 <= 2 * penwidth )
{
  left2 = left2 - penwidth
  exit /* from the segmentation procedure */
}
left2 = -1
right1 = -1
/* no segmentation possible */
exit
```

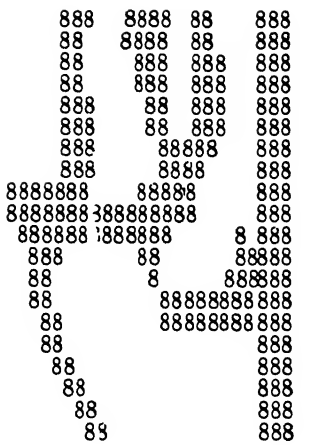
Figure 29: Algorithm for locating the right boundary of the first constituent character and left boundary of the second constituent character of a conjunct.

(a)



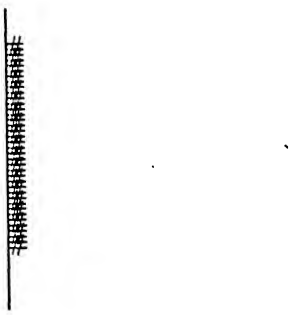
conjunct image, conj_height is 20, conj_width is 24, left_onethird is at column 8 and conj_mid is at column 12.

(d)



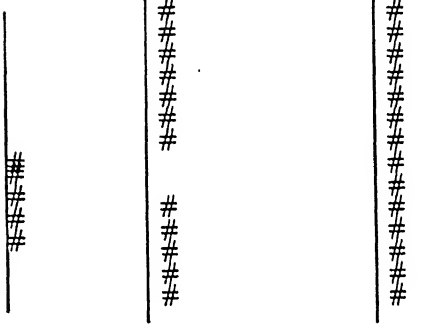
bar_pos is 21, temp_conj_right is at 2 penwidth is 3, initial value of temp_le is 18.

(b)



left_onethird has 20 rows which contain black pixels which is half of the conjunct height, left constituent character belongs class H2.

(e)

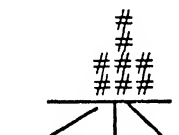


HPr at temp_left2 = 18 (connected, Hpr height 4)

HPr at temp_left2 = 16 (not connected)

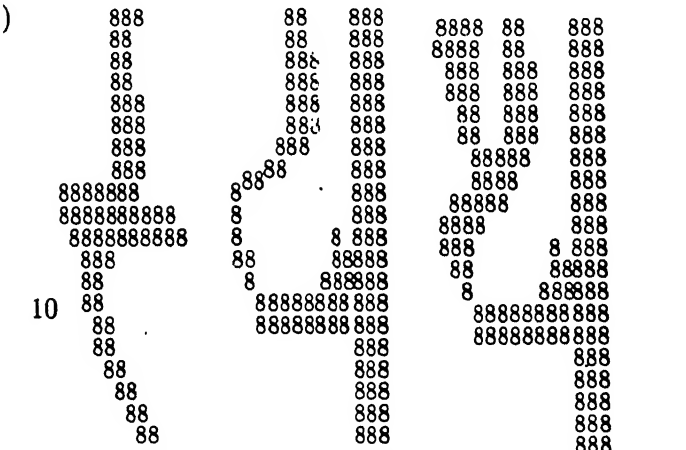
HPr at temp_left2 = 1 (connected, H height 14)

(c)



number of black pixels are same in column 8, 9 and 10. this number goes up in column 11; the value of right1 is 10. value of right1 is 10.

(f)



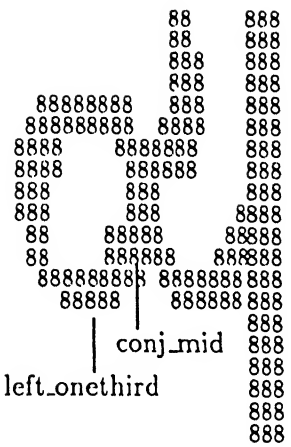
left character

right character at left2 14

right character at left2 11

Figure 30: A Conjunct and its Segmentation

(a)

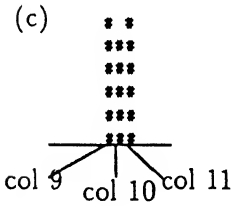


conjunct image, conj_height is 20, conj_width is 24, left_onethird is at column 8 and conj_mid is at column 12.

(b)

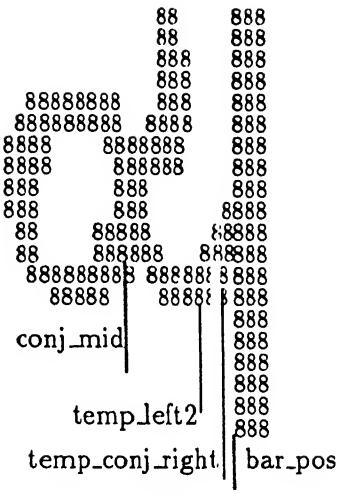
IIP11by3 has 10 rows containing black pixels which is half of the conjunct height, left constituent character belongs class III.

(c)



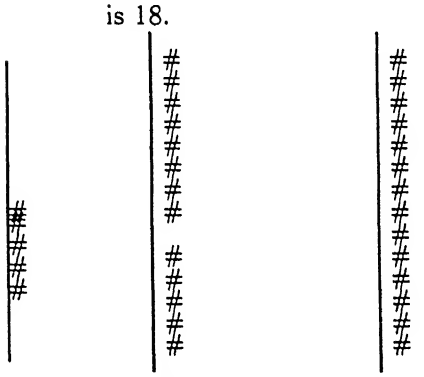
number of black pixels in column 9 is 6 which is greater than number of black pixels in column 10; number of black pixels in column 10 is less than number of pixels in column 11; the value of right1 is 10.

(d)



bar_pos is 21, temp_conj_right is at 2 penwidth is 3, initial value of temp_le is 18.

(e)

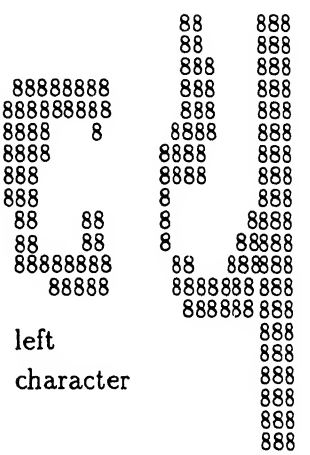


HPr at temp_left2 = 18 (connected, height 5 rows)

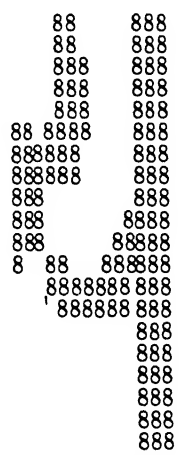
HPr at temp_left2 = 15 (not connected)

HPr at temp_left2 = 1 (connected, he 14 rows)

(f)



right charaacter at left2 = 13



right character at left2 = 10

Figure 31: A Conjunct and its Segmentation

ग घ च ज थ द ध न प फ ब भ म ल व ष स
ग्र घ्र च्र ज्र थ्र द्र ध्र न्र प्र फ्र ब्र भ्र म्र ल्र व्र ष्र स्र

This form logically corresponds to the consonant in the pure form followed by the character र (pronounced as ra). The characters which do not have a bar in the lower half, the same logical sequence is written with the help of a special lower modifier.

The base form of the character is easily obtained by removing the lower modifier. Where as, a character from its rakar form is mapped to its base form by removing the rakar modifier. Therefore, every character before classification is checked for the presence of the *slash*. In case a *slash* is present, it is removed which maps the character to its base character.

For identification of the rakar modifier, the lower half of the character box is searched for the presence of a *slash*. The following two features are used:

i. **Zero Crossings:** Horizontal zero crossing is defined as the number of crossings of a horizontal line from white pixels to black pixels for a given row of the image [5]. This feature for 'n' number of rows, yields a sequence $H \equiv h_1 h_2 \cdots h_n$ where h_i is the number of zero crossings corresponding to i^{th} row; $\overbrace{0^{th}}$ row is the top row and n^{th} row is the bottom row. In the region, where *slash* is attached to the bar, a sequence of the type '$k_1 k_2 k_3 \cdots k_m 111 \cdots 11$' is expected where $k_i \geq 2$ and the length of the sequence depends on the height of the character.

ii. **Gap in the Horizontal Projection:** Since, the slash is attached to the vertical bar, we check for the presence of a slash only the image boxes which have a vertical bar in the lower half. If the bar is present, the lower 1/3rd of the character image is explored for the presence of a *slash*. The area of interest is the image to the left of the bar. We project the image to the left of the vertical bar of the image on a horizontal line. A gap in the projection

indicates possibility of presence of a rakar modifier. This projection is also used to identify the enclosing box for the *slash* which is removed from the image.

Both the above features are checked independently; confirmation by both the features indicate the presence of the *rakar* modifier. Some of the characters before and after composition are shown figure 32.

(a) Some Consonants in their Rkar form

(b) Corresponding consonant after removing the Rkar modifier

Figure 32: Characters with Rkar modifiers and after removal of Rkar modifier.

Some of the characters take the *rakar* modifier in their shad ow. Such conjuncts are not mapped to their base form and are treated as atoms. Some of the examples are ग्र ल्र स्र .

4.8 Results

The structural segmentation algorithm has been tested on 10 printed documents. The number of conjuncts in the test documents are about 5%. Out of all the conjunct and composite characters, app. 80% of the conjunct character boxes were segmented. The fading effect violated the connectedness property of characters and the segmentation algorithm failed to find a break point where its requirement for connectedness along with other requirements were met. During the testing phase, it was observed that sometimes a composite character was substituted by

another Devanagari character. This happened when both constituent characters were relatively thin. Some of the composite characters which were not selected for segmentation are shown in figure 33.

Composite characters	Constituent charac ters	
ष्ण	ष	ण
न्ह	न	ह
स्व	स	व
स्न	स	न
न्त	न	त
न्म	न	म

Figure 33: Composite Characters not invoked for segmentation.

However, the algorithm is capable of segmenting all the conjuncts of figure 33 when suggested to do so.

The number of touching characters in test documents and number of touching characters which are segmented are summarized in tables 5 and 6 for two different fonts. The recognition performance for conjuncts is given in section 7.4.1 of chapter 7

The extracted recognition units are classified using a hybrid approach which is discussed in the next chapter.

	total number of chars.	number of touching chars.	touching chars. segmented	touching chars. not segmented
Doc I	812	38	33	5
Doc II	626	40	30	10
Doc III	527	28	15	13
Doc IV	465	26	22	4
Doc V	674	50	39	11
Doc VI	839	39	31	8
Doc VII	762	44	41	3
Doc VIII	898	46	36	10
Doc IX	756	42	42	0
Doc X	664	22	17	5
Overall	7023	375	306	69
		5.33%	81.60%	18.40%

Table 5: Number of touching characters in test documents and number of touching characters which are selected for conjunct segmentation (Font I).

	total number of chars.	number of touching chars.	touching chars. segmented	touching chars. not segmented
Doc I	1193	49	39	10
Doc II	1305	72	64	8
Doc III	1129	64	56	8
Doc IV	1368	60	52	8
Doc V	1069	60	50	10
Doc VI	605	32	26	6
Doc VII	900	46	38	8
Doc VIII	723	36	30	6
Doc IX	441	18	14	4
Doc X	637	32	27	5
Overall	9370	469	396	73
		5.00%	84.43%	15.56%

Table 6: Number of touching characters in test documents and number of touching characters which are selected for conjunct segmentation (Font II).

Chapter 5

Automatic Generation and Matching of Structural Descriptions

A structural representation has constituent strokes of a character and the relationship between them. Theoretically, the representation should remain invariant over a large number of fonts, sizes and writing styles. But in practice, a character has many representations due to the presence of noise and variations in the shape of the character. We have used structural properties of the script to guide the description generation process. In one of the earliest works [69], an attempt was made to describe Devanagari characters in terms of a certain number of primitives with their positional information encoded in the form of nine-zones of the minimum sized upright enclosing rectangular box. These descriptions were then used to perform a predictive parsing on the unknown character. These descriptions were hand-crafted. In the present work, we have automated the process of generation of description and also used certain features specific to Devanagari characters. One such feature is presence of a vertical bar. This vertical bar besides acting as a filter in reducing the set of characters over which the search is made, also gives a reference with respect

to which correspondence between junction points are established. Unlike [71] we do not obtain a piece-wise linear strokes, but permit curves of arbitrary shapes between vertex and junction points. We use a number of other gross features such as zero crossing, position of vertical bar and others to act as reliable filters, providing reduction in search space. We perform a predictive parsing on the unknown character for which a description has been obtained using the similar process as used in construction of the prototypes. In Devanagari, many of the characters are joined intentionally leading to their fusion. Similarly, there are natural breaks in the strokes due to lifting of pen. Therefore, we construct the prototypes on characters obtained through the process of segmentation on real-life words. A character class has more than one prototype if the samples during training yield distinct alternate prototypes.

5.1 Description Schema

A stroke is defined to be a segment which starts either at a *junction point* or at a *vertex point* and gets terminated at a *junction* or *vertex* point anywhere during traversal. A *junction point* is defined as one having three or more pixels of its 8-neighbours present. A *vertex point* has only one of its 8-neighbours present. The 8-neighbourhood is defined by eight neighbours of the point in north, north-east, east, south-east, south, south-west, west, north-west directions. Curvature of a stroke may change from one type to another. This change may occur many times in a specified stroke. Curvature is categorized as either convex or as concave.

Further to this definition, a *vertical bar* stroke is treated in a special manner as it plays an important role in Devanagari character description. In vertical bar stroke, we look for continuation of pixels in the vertical direction beyond junction point till a vertex point is encountered. All the junction points on this bar are treated as end points for other strokes of the character.

A rectangle box enclosing the character is divided into 9 (3x3) equal zones that are numbered 0 through 8. For each character, the position of vertical bar is recorded in

terms of zones, if it is present. Number of strokes in the character is also recorded. For each stroke, the zone in which it starts and ends is recorded. The type of points (vertex or junction) at which the stroke starts and ends are also recorded. Number of times curvature of a stroke changes and curvature type, its length and type is recorded. The information slots used by the description are shown in figure 34.

Bar **Position**
Number of Strokes
for each **Stroke**
    Position of the stroke
    Type of Begin Point
    Type of End Point
    Number of Times Curvature Changes
    for each **Curvature Change**
        Nature of Curve
        Normalized Length of Segment

Figure 34: Description Schema.

5.2 Generation of Description

The character box is divided into three vertical segments of equal width. Each segment is independently checked for the presence of a vertical bar. For Devanagari characters, vertical bar can appear only in the middle or right segment. The segment in which the bar lies is recorded as *barPos*. *barPos* is either *END_BAR*, *MID_BAR* or *NON_BAR*.

If a bar is found, all junction points on the bar are recorded and it is removed from the image.

The stroke starting at the left most vertex point or junction point is extracted next. This point is deleted from further consideration. Since Devanagari and its characters

. are written from left to right, the left-most point is a natural choice. The description of the stroke is added to the character description.

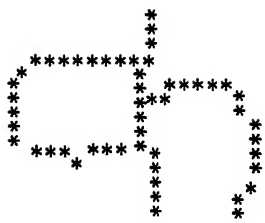
This process is repeated till all the recorded points are exhausted. Using the above data structure and algorithm, we generate stroke description for all the characters of the Devanagari script.

In figures 35 and 36, strokes of two characters and their descriptions are shown.

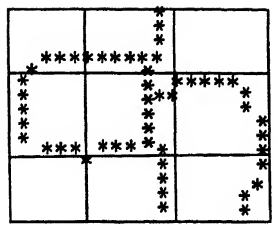
It may be noted that special treatment of the bar leads to natural descriptions of characters which have a bar either at the end or in the middle. If the vertical bar is treated like other strokes, the bar is broken into small strokes at all junction points. An obvious modification is to let a stroke continue as long as it could. But this does not solve the problem. The descriptions are still far from natural. Figure 37 shows two characters with bars and strokes obtained by using above three definition of strokes.

As pointed out earlier, there are natural fusions of characters in the form of conjuncts which are attempted to be separated. As no segmentation algorithm can yield clean boundaries, it is imperative that the prototype descriptions are generated using real life patterns rather than being hand-crafted using ideal patterns. Secondly, characters in Devanagari script may also have natural breaks. Thus, the prototype descriptions generated are natural and capable of taking natural variants into account. *Small* sized strokes are treated as noise and neglected. A stroke of a character whose length in less than 1/3rd of the minimum of length and width of the character is a *small* stroke and ignored as noise. The length threshold is set for every character when strokes are extracted. Different descriptions of the same character are stored as alternative descriptions of the character. The description generation algorithm is given in figure 38. Bit map of the skeleton is passed to the description generator. All the junction points and vertex point are put in an array called *pointArr*. Initially, *pointArray* will contain only *vertex* points if no bar is present.

(a)

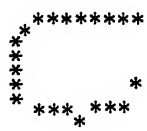


(i) Character skeleton



(ii) Character skeleton with enclosing box

(iii) Mid-bar



(iv) Stroke 1

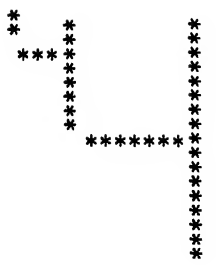
(v) Stroke 2

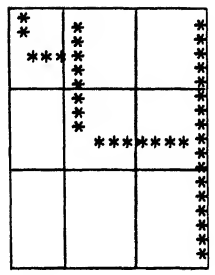
(b) Values of various slots of the description schema for the above character

barPos	MID-BAR	
numStrokes	2	
stroke 1		
	beginBoxNum	1
	typeBeginPoint	JUNCTION_POINT
	endBoxNum	4
	typeEndPoint	JUNCTION_POINT
	numof CrvChg	1
	typeofCurve	CONCAVE
	lenOfSeg	17
stroke 2		
	beginBoxNum	4
	typeBeginPoint	JUNCTION_POINT
	endBoxNum	8
	typeEndPoint	VERTEX_POINT
	numof CrvChg	1
	typeofCurve	CONVEX
	lenOfSeg	10

Figure 35: Strokes of character क and values of slots of the description schema as obtained by description generation process.

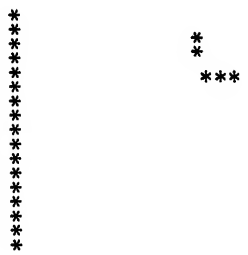
(a)





(i) Character skeleton

(ii) Character skeleton with enclosing box



(iii) End-bar (iv) Stroke 1 (v) Stroke 2

(b) Values of various slots of the description schema for the above character

barPos	END_BAR	
numStrokes	2	
stroke 1		
	beginBoxNum	0
	typeBeginPoint	VERTEX_POINT
	endBoxNum	1
	typeEndPoint	JUNCTION_POINT
	numof CrvChg	1
	typeofCurve	CONCAVE
	lenOfSeg	6
stroke 2		
	beginBoxNum	1
	typeBeginPoint	JUNCTION_POINT
	endBoxNum	5
	typeEndPoint	JUNCTION_POINT
	numof CrvChg	1
	typeofCurve	CONVEX
	lenOfSeg	14

Figure 36: Strokes of another character ਮ and values of slots of the description schema as obtained by description generation process.

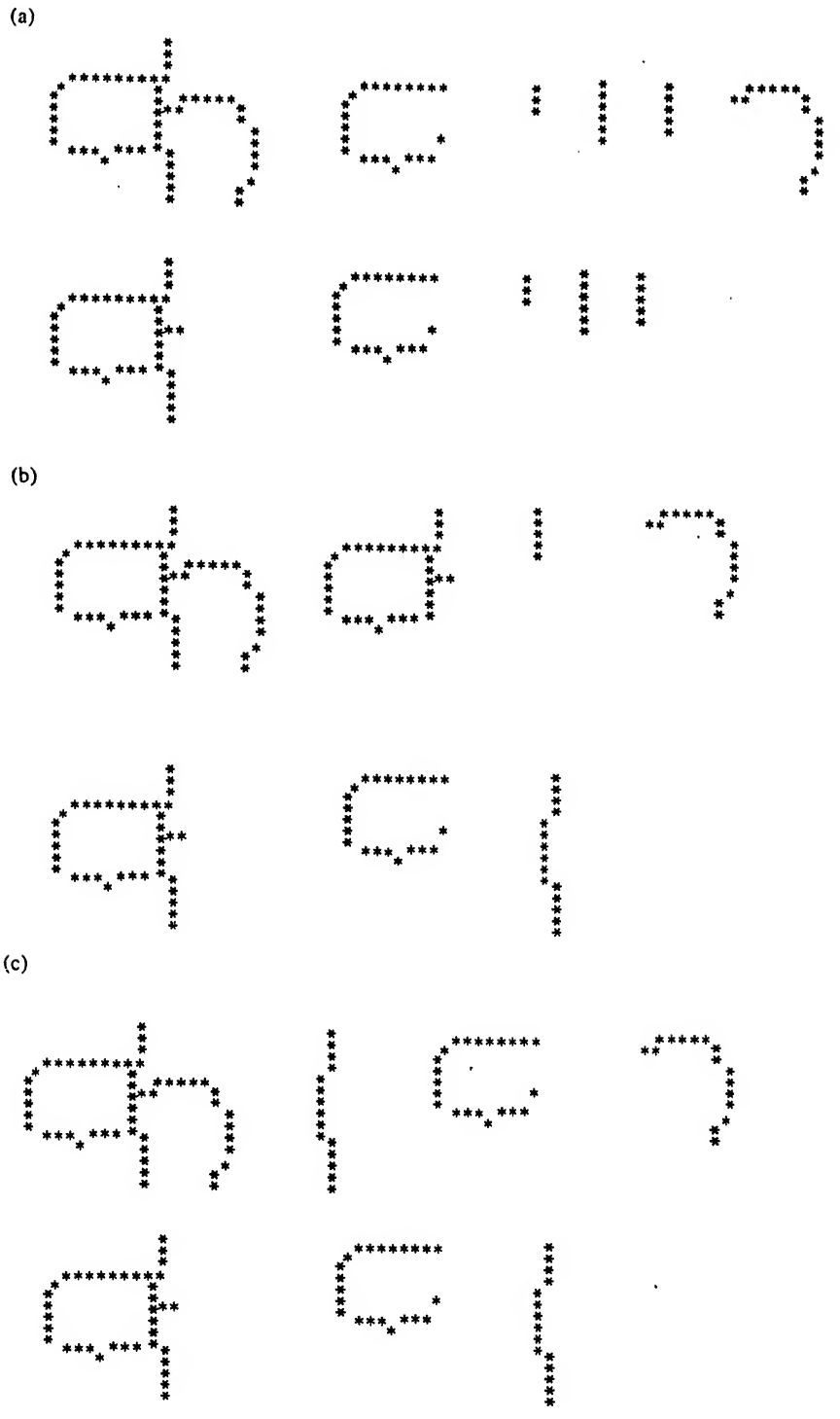


Figure 37: Descriptions obtained using three different definitions of a stroke; (a) Descriptions of characters on the left, when a stroke is terminated at a junction point; (b) Descriptions of characters on the left, when a stroke is allowed to continue as long as it could; (c) Descriptions of characters on the left, when the vertical bar is treated as a single stroke and all other strokes terminate at a junction point.

```
Input: Image
Output : All slots of REP filled up
Algorithm:
  0. Set stroke length threshold to 1/3rd of minimum
       of character height and width
  1. Locate and add all vertex points to pointArr
  2. Divide the Image into 3 ve tical segments
     Locate the segment with vertical line
     if vertical line is present
       a.   Fill barPos slot of REP
       b.   Remove the vertical line from the Image
       c.   Add the junctions points found on the vertical line to pointArr
       d.   Remove vertex points of vertical line from pointArr
  3. Choose a point PT from pointArr
     Fill beginBoxNum and typeBeginPoint slots
  4. Remove the PT from pointArr
  5. Start traversing the stroke from  PT
     if a junction point is reached
     a. pick the neighbour in the diction of least change as next point
           of the stroke
     b. add all other neighbours to pointArr
  6. At the end of traversal of this stroke,
     if stroke length is more than the threshold
     a.  Fill endBoxNum and typeEndPoint slots
     b.  increment numStrokes by 1
     c.  Analyze curvature of the stroke
         Fill numTypeofCrvChg slot
         for each change
            Fill typeofCurve and lenofSeg slot
  7. Remove end point from pointArr
  8. if pointArr is empty, stop
  9. GO to step 3.
```

Figure 38: Algorithm for Generation of Descriptions.

5.3 Matching Process

Before applying the matching algorithm, a preliminary classification is performed. We use presence and position of vertical bar to classify the characters into three classes, the END_BAR characters are further divided into two classes based on the nature of the joining pattern with the header line. The horizontal zero-crossing feature is applied next for further reducing the number of candidate characters. These features have been described in section 2.6.1 and 2.7.1 of chapter 2.

Now the unknown character is matched with the set obtained after preliminary classification. The matching is expectation driven and done in an incremental way. If two descriptions do not match at the first level, i.e. number of strokes and their positions, an effort is made to transform the description of unknown character to match the prototype. A mismatch incurs a penalty on the prototype. The transformation process is guided by the positional information of the constituent strokes. Two strokes are joined together if the two points under consideration for joining are in the same zone of the character. Two strokes are matched by checking relative positions of their *begin* and *end* coordinates. If the relative positions do not match, a penalty is given. A comparison is made for the types of begin and end points to see whether they are junction points or vertex points. A mismatch in the type incurs a higher penalty than the mismatch in relative position. The types of the curve is also compared. A mismatch in the types of curves incurs a higher penalty than mismatch in relative positions of strokes because it is unlikely that the curvature of a character will change even if font or writing style has changed. The penalty for mismatch in segment lengths is obtained by finding the difference in segment lengths and multiplying it by a weighting factor. The algorithm is given in figure 39.

As mentioned earlier, the algorithm which generates prototype description is also used for generating description for the unknown character. If number of strokes are same, strokes of the unknown character and that of prototype can be matched sequentially. No reordering of the strokes is required.

```
1. Initialize Cost to 0
2. match number of strokes for equality
   if match fails, go to step 5
3. for each stroke of prototype
   a.  match relative position of end and start points
   b.  match type of end and start points
   c.  match number of curvature changes
   d.  for each segment
          match type of curve
          match normalized segment length
   /* for each mismatch, add preset penalty */
   /* to Cost                                */
4. stop

5.  Cost = difference in number of strokes * preset_weight

6. /* If prototype has more strokes than the description of */
   /* unknown character,                                    */
   /*   strokes of prototype are joined                     */
   /* else                                                  */
   /*   strokes of unknown character are joined             */

 If prototype has more strokes than the description of
   unknown character,
     supergraph = prototype
     subgraph = unknown character description
   else
     supergraph = unknown character description
     subgraph = prototype

7. For each stroke of subgraph
   a.    a stroke in supergraph is located with matching beginning
   b.    another  stroke in supergraph is located to be joined with
         the selected stroke
   /* for each match, subtract a preset number from the Cost */
8. Stop
```

Figure 39: Algorithm for matching the description of an unknown character with a stored prototype of a candidate character.

5.4 Illustration

Thinned image of true character य which is pronounced as *ya* is considered as an example. The description for this image as generated by the corresponding module is given in figure 40.

barPos	END_BAR	
numStrokes	1	
STROKE 1:		
	beginBoxNum	1
	typeBeginPoint	VERTEX_POINT
	endBoxNum	5
	typeEndPoint	JUNCTION_POINT
	numofCrvChg	2
	typeofCurve	CONVEX
	lenOfSeg	5
	typeofCurve	CONCAVE
	lenOfSeg	8

Figure 40: Description of the Sample Character य.

Position of the vertical bar is compared with the prototypes. The prototype class which has characters at the end is selected. Horizontal zero crossing filter is also applied to eliminate some more characters from the class. A set which contains 4 candidate characters is obtained. These four characters are म य भ अ. The structural representations of these characters as stored in prototypes are shown in fig 41. Notice that म and अ have more than one descriptions.

In this example, number of strokes of the unknown does not match with the first prototype of character म, the first character of the candidate set. A transformation is

proto. no.	1	2	3	4	5	6	7
true char	म	म	म	य	भ	अ	अ
barPos	D	D	D	D	D	D	D
numStrokes	2	1	1	2	1	2	2
stroke1							
bgnBoxNo	0	0	0	1	0	1	1
typBgnPt	V	V	V	V	V	V	V
endBoxNo	3	5	4	3	5	5	5
typEndPt	J	V	V	J	V	V	V
nofCrvChg	1	1	1	1	2	1	1
typofCrv1	X	E	E	E	X	E	X
lnOfSeg	8	13	12	6	6	7	13
typofCrv2					E		
lnOfSeg					8		
stroke2							
bgnBoxNo	3			5		4	4
typBgnPt	J			J		V	V
endBoxNo	5			3		6	3
typEndPt	J			J		V	V
nofCrvChg	1			1		1	1
typofCrv1	X			E		X	X
lnOfSeg	7			6		9	9

Figure 41: (Legend: J : Junction Point, V: Vertex Point, X: Convex Curve, E: Concave Curve, D: End Bar) Descriptions for the Candidate Characters.

done and matching continues. The *beginBoxNum* does not match. The *typeofCurve* of second stroke does not match. This incurs a penalty on the prototype. With second prototype of the same character, number of strokes matches, relative position of begin and end points matches. Type of begin point matches but the type of end point did not match. In the prototype, the end point is a junction point. In the description of unknown character, it is a vertex point. This incurs a small penalty. The type of curves do not match. With the third prototype of the same character, the number of strokes matches; the relative position of stroke matches but the nature of curve does not match. Fourth prototype is transformed by merging two strokes into one. The relative position of strokes, type of curve both matches. The matching process continues and the minimum distance character is found to be य which is also the true character.

5.5 Results

We have experimented with the system on printed documents. We trained the system on more than ten document pages containing more than 12,000 characters. The number of prototypes generated varied from 12 to 25 for a single character class, the average being 13. The number of different prototypes for 70% of the character classes was 5 or less and 10 for 15% character classes. However, the remaining 15% character classes the number of prototypes was 25. In approximately 70% cases, output came out to be an unique true character. In approximately 18% cases, the true character is among the next two choices and in 5% cases the true character was in the subsequent two choices. These results have been summarized in table 7. Certain character classes could not be discriminated, primarily because all *short strokes* have been ignored. A stroke is *short* if its length is less than the height of one 3*3 zone. Word level knowledge is able to resolve most of the confusions. However, we have also employed a character pair expert to look more closely at the image whenever required. Some of the confusion classes are shown in figure 42.

(a)	ल	ठ	प	प	स	भ	द
(b)	त	ढ	य	म	म	म	ट

Figure 42: Penalty figures for character in row(a) is same as for the character in row(b) for each column, when the true character is either of the two.

The characters ल and त constitute approximately 6% of the text on the average. Whereas, प, य, म, स and भ constitute roughly 7% of the text. However, the substitution errors were approximately 9% while the rejection errors were less than 1.0%. However, 90% recognition rate is achieved after the post-processing stage which uses word level knowledge. In the next chapter, we describe the post-processing phase.

		RECOGNITION				ERROR	
	number of chars.	top choice	in next two choices	in subsequent two choices	in remaining choices	substitution	rejects
Doc I	756	541	147	28	16	21	3
Doc II	631	436	105	31	24	34	1
Doc III	730	539	117	39	18	15	2
Doc IV	752	544	126	36	22	24	0
Doc V	897	620	159	44	24	45	5
Doc VI	605	380	129	25	32	39	0
Doc VII	450	293	91	13	20	33	0
Doc VIII	723	530	122	36	17	16	2
Doc IX	441	280	90	13	23	33	2
Doc X	637	409	120	28	23	53	4
Overall	6622	4572	1206	293	219	313	19
		69.04%	18.21%	4.42%	3.30%	4.72%	0.28%

Table 7: Performance of the system at character level for Font I.

	number of chars.	RECOGNITION				ERROR	
		top choice	in next two choices	in subsequent two choices	in remaining choices	substitution	rejects
Doc I	1193	793	175	73	26	124	2
Doc II	1305	901	196	78	35	92	3
Doc III	1129	729	134	63	58	127	18
Doc IV	1369	971	173	80	32	108	4
Doc V	1069	757	129	66	23	90	4
Doc VI	605	417	91	36	16	40	5
Doc VII	900	598	130	55	20	96	1
Doc VIII	723	481	104	44	16	77	1
Doc IX	441	291	62	26	10	50	2
Doc X	637	452	76	38	13	54	4
Overall	9370	6390	1270	595	249	858	44
		68.19%	13.55%	5.96%	2.65%	9.15%	0.46%

Table 8: Performance of the system at character level for Font II.

Chapter 6

Word Hypotheses Generation and Correction using Dictionary

The most common mode of interaction with computer is through keyboard. Isolated optical character readers (IOCRs) and speech recognition systems provide alternate modes of interaction. In every mode, the errors in the input are made either due to human mistakes or limitations of the software systems. Many spelling checking programs [59] are available for detecting these errors. There are two approaches for judging the correctness of the spelling of a word. One estimates the likelihood of a spelling by its frequency of occurrence [31, 55, 62, 63, 83, 88] which is derived from the transition probabilities between characters. This requires a priori statistical knowledge of the language. In the other approach, the correctness is judged by consulting the dictionary. A hybrid [31, 83, 87] approach attempts to combine the best of both the approaches by amalgamating them at an appropriate stage of processing. The English optical text readers have employed contextual processing extensively [71, 72, 87] for improving the performance.

The spelling correction programs also offer suggestions for correct words which are based on the similarity with the input word using the word dictionary. Here, a mechanism is required to limit the search space. A number of strategies have been

suggested for partitioning the dictionary [73] based on length of the word, envelop and selected characters. In practice, a combination of these strategies is used to ensure the selection of the right partition in spite of certain classification errors [89] in the input word. In case of IOCR, the nature of errors depend on the classifying features used. There are certain types of errors which are more frequent. For instance, IOCR may have difficulty in disambiguating between character classes (e, c), (u, v) and (g q). These errors are unlikely in keyboard entry as characters of each pair are located on different rows of the keyboard. Therefore, the candidate word selection process must know the nature of errors and make use of this knowledge while making candidate word selection. In case of IOCR which is the concern of this thesis, this knowledge consists of confusion classes which are learnt through training/testing. Takahashi et al [89] characterize and classify a word according to a constant number of characters selected from all the characters of the word. However, they ignore the relative position of the selected characters in the word. As a result, inappropriate words are selected as candidate words. This leads to extra work and puts responsibility on the similarity matching process to filter out inappropriate candidate words.

The similarity matching function must also use the confusion matrix to assign penalty for a mismatch between character pairs. Instead of assigning the penalty uniformly as in [89], we suggest a penalty matrix which contains different penalties for different types of mismatches. The penalty matrix assigns less penalty to a mismatch when it is a known confusion (see section 6.5 for details).

The correction method presented here uses a partitioned Hindi word dictionary. The partitioning scheme has been designed keeping special problems in mind which Devanagari script poses. Devanagari script is a two dimensional composition of symbols and characters which requires segmentation in vertical as well as in horizontal direction; fusions and fragmentations are natural. Our Hindi[1] dictionary words at top level have been divided into two partitions, namely: short words partition and the long words partition. In Hindi dictionary, more than 30% words contain one

[1]Hindi is official language of India which is written in Devanagari script.

or two characters excluding the modifiers. We refer to these words as *short* words and all other words as long words. The envelop information consisting of number of *top, lower, core* modifiers along with number of core characters forms a partitioning feature for further partitioning for short and long word partitions. In addition, each long word has finer details of its envelop associated with it in the form of a vector. The words of each of these partitions are then replicated into various sub-partitions based on *tags*. This partitioning facilitates generation of word hypotheses based on the word envelop with coarse character level recognition. A *tag* is a string of length two associated with each partition. A word is included in a partition if it contains the characters of tag in the same relative sequence. This way some redundancy does get introduced. However, this redundancy ensures that if any two characters of the word have been recognized correctly, at least one tag will point to the correct partition.

An input word is searched in the selected partitions of the dictionary. An exact match stops further search. However, while looking for an exact match, the best 3 choices are gathered. The ranking of the words is based on their distances from the input word. The distance measuring criterion is described in section 6.5. If the best match is within a preset threshold distance, further search is terminated. However, for *short* words, no search terminating threshold is used. Instead, we try various aliases which are formed from the output of classification process. The output of the character classification process is of three kinds:

1. a character is classified to the true class - correct recognition;
2. a character is classified such that the true class is not the top choice - substitution error;
3. the character is not classified to a known class - reject error.

Kahan [39] et al use the confusion matrix to generate aliases. First alias is generated by substituting first input character by its known confusion and each character is substituted in the same way for generating other aliases. The cost of an alias is the probability that an input character is classified as its known confusion. In the second round, additional characters are substituted and the cost is revised. These aliases are searched in the dictionary till an exact match is found or the cost of alias

is too high. They assume that the true word will always be one of the aliases and an exact match for the alias is searched. This assumption is not realistic because the rejected characters will have to be substituted from the dictionary. Moreover, a substitution error and its confusions may not contain the true character which again require mappings from the dictionary.

In case of substitution errors, the true character may be missing or may be present in the set of alternate choices. We form aliases from the top three choices for each character assuming that the true characters of a word are either the top choice or present in the subsequent two choices. In case, the true character is missing from the set of choices, the substitution errors are corrected by mapping. We use confusion matrix and gross features (see section 6.5) for selective mapping of a character. For *short* words, the number of aliases formed is not large and we try each alias till an exact match is found or all the aliases are exhausted. For other words, the number of aliases may become large depending on the number of characters in the words. To avoid searching all the words, we set a threshold on the distance and terminate the search when a word within the threshold distance is found. We use only top choice for the modifiers while forming the aliases.

In case of Roman script, a word consists of the classified characters. The word is then verified and corrected if necessary with the help of a dictionary or by using the statistical knowledge of the language. Whereas in case of Devanagari script, a word is segmented into symbols and characters for the purpose of classification. Therefore, a composition phase is required to compose back the word from its constituent units. This phase is an integral part of a Devanagari document reading system. All units of a word are composed back by a composition processor using Devanagari script composition rules into a syntactically valid Devanagari word. If a unit is classified into more than one class, a word for each class is formed. It is possible that some of the classes do not form a syntactically valid word. The composition processor tries to correct the word with composition rules. Composed words are then verified and corrected by a post-processing phase using a word dictionary. In the next section, composition phase is described. Partitioning scheme is described in section 6.3.1

and matching process is also described in section 6.5. The experimental results have been presented in section 6.6 followed by concluding remarks in section 6.7.

6.1 Character Composition Phase

The composition phase does two things:

1. associates the modifier symbols with core characters/symbols,
2. composes all units of a word to arrive at a syntactically meaningful word.

6.1.1 Association of the Modifier Symbols

The modifiers may be broadly classified into three classes:

left-going modifiers a top modifier is left-going if the point where it touches the header line is almost at the right end of the modifier. A lower modifier may not touch a core character. Therefore, the highest point of a lower modifier is used as a reference point. A lower modifier is left-going if the highest point of the modifier is almost at the right end of the modifier. Some of the left-going modifiers shown attached to the character क are the following: कै के कु की.

right-going modifiers a top modifier is right-going if the point where it touches the header line is almost at the left end of the modifier. A lower modifier is right-going if the highest point of the modifier is almost at the left end of the modifier. Some of the right-going modifiers shown attached to the character क are the following: कॅ कि कू.

centrally located modifiers a modifier (lower as well as upper) is centrally-located if the modifier and its carrying core character both are vertically aligned. Some

centrally-located modifiers shown attached to an appropriate character are the following: ट्र कं

In case of centrally located modifiers, association with core characters is simple as there is vertical overlap between the modifier and its carrying core character. For other cases, let us first see the nature of association. In Devanagari script, a core character touches the header line at least at one point. In some cases, a character may touch header line at two points. The right most touching point is used for placing an upper modifier. Association is simple if the modifier is left-going and it has been placed at the right end of the core character image. If a modifier is placed at the center of a core character, association is again easy. The following words are easy to compose as the modifiers are of the above described type and there is sufficient overlap between the modifier and the core character:

मुझे अपने अनुभव हैं

If a modifier if right-going and it is placed at the right end of the core character, there is hardly any vertical overlap. This is true for lower modifiers as well as for upper modifiers. We know that the vertical bar । is also a valid character. This character is thinner than any of the upper modifiers that can be placed with it (no lower modifiers can be attached to it). Some of the difficult cases are are shown below:

मिली वर्ष मोर अनुकृति भार्या

In all these cases, the vertical overlap is very little between a core character and its modifiers. It is obvious that vertical overlap cannot be used for association. However, in all these cases, beginning of the modifier is to the left of the end of the core character. We use this observation to formulate the following association rule: *associate a modifier with the left most core character that ends after the modifier starts.* Some example associations as obtained by above rule are shown in figure 43.

(a) Association of modifiers with core characters of word मिली:

(b) Association of modifiers with core characters of word मोर:

(c) Association of modifiers with core characters of word अनुकृति:

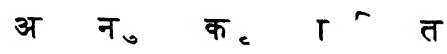


Figure 43: Association for three different words using the association rule.

These associations are correct for all modifiers except when a ा is involved. These errors are corrected by using the syntactic rules of the Grammar as explained next.

6.1.2 Composition

After associating the modifiers with core characters, the units are composed together. If all classifications and associations are correct, composition is trivial. In the presence of errors, composition may fail. Input to the composition processor is the output of recognition process. It is possible that a character is classified to more than one known class. In other words, for an input character, multiple outputs are produced by recognition process. We use every alternative of each character produced by recognition process to form words. Therefore, corresponding to one input word to the text reading system, multiple words are formed by the composition processor. Some of these words may be identical. Here is an example:
input word: हमारा
OCR output: (ह, द) (म, भ) (ा) (र, द) (ा)
Symbol strings: हमारा दमारा हभारा दभारा हमादा दमादा हभादा दभादा
Each symbol string is checked for syntactic validity. There are several rules [71] which have been carefully formulated based on the syntactical rules for the script. These rules guide the composition processor in identifying the symbol sequences

that are syntactically correct. Some of the composition rules formulated are shown in figure 44.

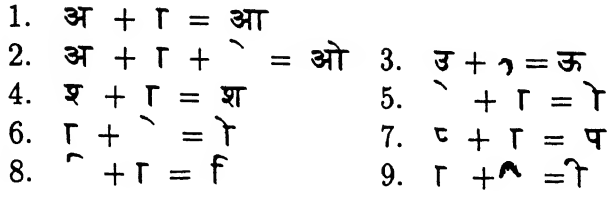


Figure 44: The composition rules for Devanagari Script.

Getting back to our example word of figure 43(a), word मिलो is composed by using rules 8 and 9. मोर, character म is followed by two modifiers: ◌ ा ; rule 5 is applied that composes both the modifiers into ो to produce correct word मोर. Rule 8 is used to compose the word अनुकृति.

There are situations when an invalid string cannot be corrected. A reject or substitution error at symbol level may make a symbol string invalid. Consider the word मिलो again. OCR output for this word is the following:

input word: मिलो

OCR output: (ा, र) (म, भ) (◌ि) (ल, त) (ा) (◌ो)

character ा has also been classified as र. The string र म ◌ि ल ा ◌ो is an invalid string. Such un-composed strings are also passed to the verification process. The verification process uses confusions of characters and tries to find a close match.

6.2 Correction Using Dictionary

The correction process checks the given word in the dictionary. If the word is present, no more processing is done. In case, the word has some unclassified characters, the

correction process selects candidate words from the dictionary using the classified characters of the word. The unclassified image boxes are substituted by a character in a constrained manner. The purpose is not to correct typographical errors as in case of a spell checker. There, alternative words for misspelled words are also suggested to help the user who may decide to retain the original word or may modify it. For example, when a spell checker gets *begm* as its input, possible alternatives are be beg, begging, begin, beggar etc. But for a text recognition system, the only alternative is *begin*. This error has resulted from the fusion of *i* and *n*.

For Devanagari script, these constraints come from the structural properties of the script and substitution errors which the classification process is known to make. Position of lower and upper modifiers also play a vital role in selection of words from the dictionary. The vertical bar property is not violated during substitution.

Some of the valid and invalid substitutions are given below for true word आदर:

True word: आदर
Composed word: घादर
Allowed substitutions: चादर आदर सादर
Invalid substitutions: घातक आकर आदत

We next explain the partitioning strategy and creation of the dictionary followed by selection process.

6.3 Dictionary Organization

Our dictionary contains more than 21,000 words. The dictionary needs to be partitioned in order to reduce the search space besides preventing forced match to incorrect word. The partitioning strategy should be such that the true word corresponding to the input word is always found in the partition possessing the same partitioning feature as the input word. The input word possesses the same

feature as the true word as long as the number of errors are below a certain threshold. Our hierarchical partitioning strategy is based on the word envelop and character tags.

6.3.1 Partitioning of the Dictionary

The envelop of a word contains the following information:

1. number of character boxes
2. number of vertical bars
3. number of upper modifier boxes
4. number of lower modifier boxes
5. vector giving position of vertical bars
6. vector giving type and position of each character box

The number of character boxes excluding the modifiers are used for partitioning at the top level. The number of lower and top modifiers are used at the next level. We do not use the position of the modifiers because the span of modifiers does not always correspond with the characters being modified. As a result, the position information of the modifiers is not reliable. In addition, number and position of core modifier - vertical bar: | is used. This core modifier has a very high frequency of occurrence. We have found from our experimentation that the recognition rate for this modifier symbol is very high. The partitioning feature is the number of core characters, top, lower and core modifiers. For two core character words we allow at most two modifiers of each type. Thus, we require 27 partitions to incorporate all the combinations. Similarly, we need 8 partitions for single core character words while number of modifiers of each type varies from zero to one. Out of all these

combinations, some of the combinations of modifiers do not form valid syntactic words. For the valid combinations, we require only 28 partitions which are listed in table 9. Sometimes, number of modifiers of one or more types is more than one in case of one core character words and more than two in case of two core character words. However such words are not many and we put all of these in a single partition. Table 9 gives the number of words in each partition. The average number of words in a *short* word partition is 146, minimum is just 1 and maximum number of words in a partition are 781.

number of core characters = 2							
no. of modifiers			no of	no. of modifiers			no. of
lower	top	core	words	lower	top	core	words
0	0	0	242	0	0	1	171
0	0	2	7	0	1	0	302
0	1	1	114	0	1	2	5
0	2	0	98	0	2	1	16
1	0	0	388	1	0	1	170
1	0	2	12	1	1	0	781
1	1	1	166	1	1	2	6
1	2	0	338	1	2	1	70
1	2	2	1	2	0	0	225
2	0	1	22	2	1	0	453
2	1	1	40	2	2	0	314
2	2	1	25				
number of core characters = 1							
0	0	1	62	0	1	1	66
1	0	0	12	0	1	0	27
1	1	0	10				
number of core characters = 1 or 2							
the remaining combinations of lower, top and core modifiers not covered above							236

Table 9: Partitions and number of words in each partition for *short* words based on the word envelop and selected characters partitioning feature.

Long words are first partitioned using the same feature as in case of short words. In addition, a vector indicating the positional information of vertical bar and each core character along with the type of each character. The possible values for type of the box is '0' for NON_BAR characters, '2' for MID_BAR and '3' for END_BAR characters. The vertical bar is represented by '4' at the same index position in the vector as in the word. For instance, consider word भारतीय. The vector '040040' indicates position of vertical bars. The vector '2*02*2' indicates the position and type of each core char where '*' represents vertical bar. Since the position of vertical bar and core characters are disjoint, these two vectors are combined together. The combined vector is ' 240242'.

The long words of each sub-partition are replicated into various groups with the help of *tags*. A *tag* is a string of characters associated with a partition. A word is added to a partition if the word contains the characteristic tag associated with that partition. The length of the tag is 2 characters, therefore, a word of length 4 has 6 tags; all of which may not be distinct. For instance, the word भारतीय has six distinct tags: भर, भत, भय, रत, रय, तय. Therefore, the word भारतीय goes into six partitions which correspond to these tags. Figure 45 shows six partitions and words of each partition. This way some redundancy is introduced. This redundancy ensures that if any two characters of the word have been recognized correctly, at least one tag will point to the correct partition. If the character classification process (IOCR) has a recognition rate of 70%, two characters out of three on the average will be correctly classified. Consequently, for a three character word, out of three tags, one will correspond to the partition containing the true input word. The required tag length at the same performance remains same for four character words. Our experimentation has shown that approximately 65% words contain two core characters, 18% contain three core characters and the remaining words are of length four or more. In other words, tag length two is necessary if the recognition rate is in the neighbourhood of 70%. Number of partitions, repetition count etc. for 10500 long words are shown in figure 46. The hierarchical structure of the dictionary is represented in figure 47.

tag	words in the dictionary partition
भर	भारतीय गम्भीर भारीपन भोगकर भोजराज उभारती
भत	भारतीय प्रभावित भानमती भागवती सभापति संभालता उभारती गम्भीर
भय	भारतीय अभियान
रत	भारतीय कठोरता प्रसारित शोहरत स्थिरता उभारती उतारती
रय	भारतीय नारियल फरियाद रियासत
तय	भारतीय आतिथ्य साहित्य फरियाद रियासत

Figure 45: A few tag partitions and words of each partition.

Total number of words	10,500
Number of Partitions	2841
Total Number of words in 2841 partitions	73211
Average Number of repetitions for a word	7

Figure 46: Statistics of Partitions

6.4 Hypotheses Generation

The partitioned dictionary is used for generating hypothesis for each character box of a word. The envelop information is extracted from the true word image which is used for selecting a dictionary partition. In case of long words, the vector associated with the true word is also compared with the dictionary word. The word is selected only if the two vectors match. The hypothesis set for each character box is constituted by corresponding character of each word of the selected partition with matching tag.

The set of candidate characters constructed by gross classification features (see chapter 2) and hypothesis set constructed from dictionary are compared. Only the common elements are retained as the revised set of the candidate characters. The empty revised set for a character box provides a clue to the segmentation process to reconsider its segmentation (see figure 48).

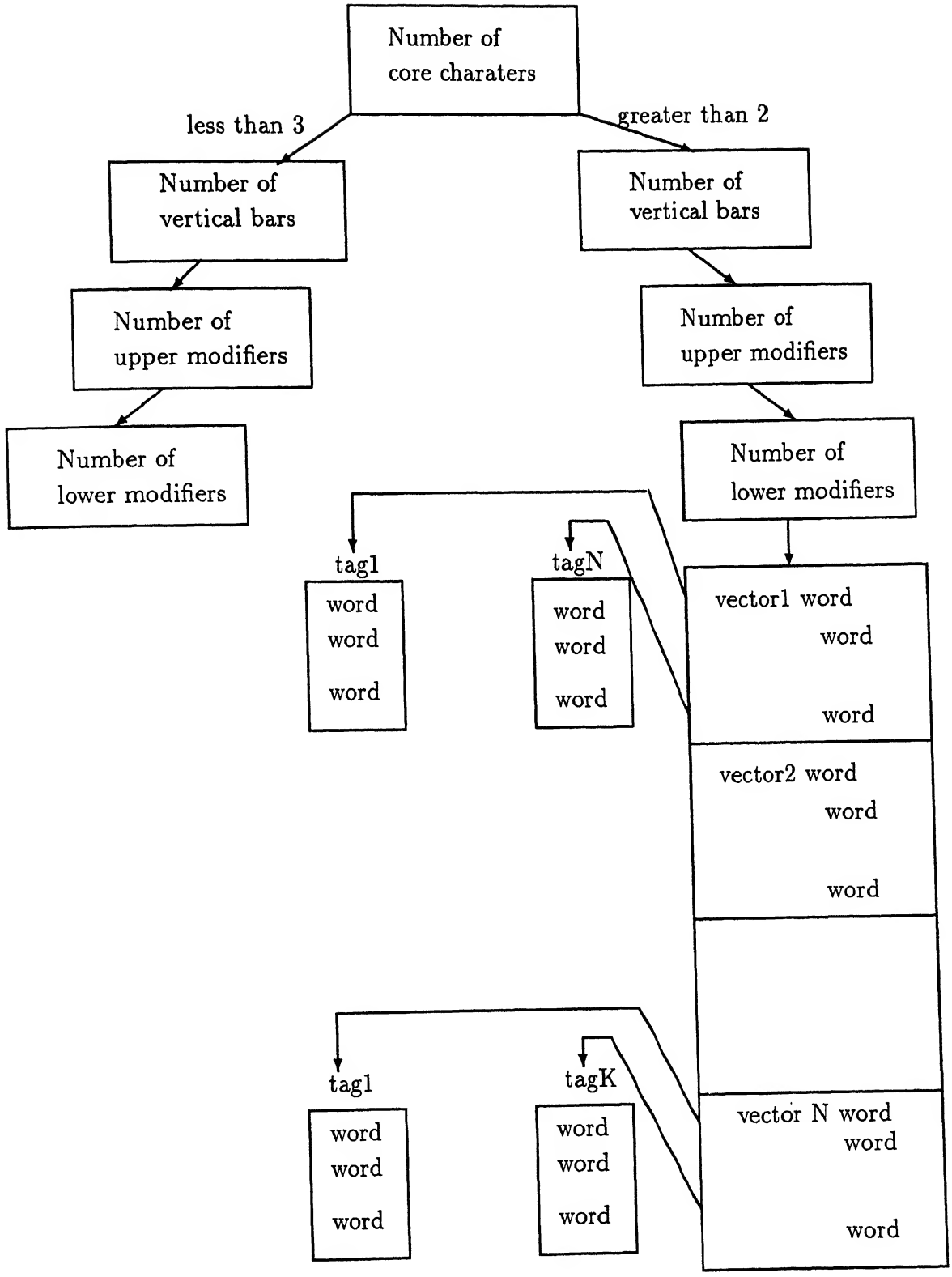


Figure 47: Dictionary Organization For Devanagari Text Recognition System.

True word: स्तर
Initial segmentation: स्त र
Initial word envelop:
number of character boxes 2
number of vertical bars 0
number of upper modifier boxes 0
number of lower modifier boxes 0
Word hypotheses based on word envelop, vertical bar property and junction with header line of each character box:
पर भर मर यह घर खट फट फर पट पद सह टर
Corresponding character box hypotheses box1: प भ म य घ ख फ स ट
box2: र ट द
Character hypotheses supported by Modified Horizontal Zero crossing vector:
box1 : none
box1 is a wide character
Resegment box1
refined segmentation: स्त र
Modified word envelop:
number of character boxes 3
number of vertical bars 0
number of upper modifier boxes 0
number of lower modifier boxes 0
Word hypotheses based on word envelop, vertical bar property and junction with header line of each character box:
स्तर स्वर
Corresponing character box hypotheses box1-1: स्
box1-2: त व
box2: र
Character hypotheses supported by Modified Horizontal Zero crossing vector:
box1-1 : स्
box1-2 : त ल य
box2 : र
candidate word: स्तर

Figure 48: Example showing word hypotheses generation process

6.5 Matching

We assume that the character units decomposed by the segmentation process have been composed back in valid syntactic character strings using Devanagari script composition grammar[71]. The modifiers have been placed appropriately with their carrying characters.

Two main sources of errors are incorrect classification and segmentation. The fusion and fragmentation of characters alter the length of a word with an exceptional situation when change caused by fusion is offset by a fragmentation. The errors caused by fragmentation and their correction has been dealt with by Sinha in [71]. The fusions in Devanagari script are mandatory as a character in its *half* form always touches the following character. The lower modifiers are mostly written touching the carrying character. The fused character boxes are segmented into its constituent characters. Irrespective of the mechanism used for selecting the fused character boxes for further segmentation, two types of errors can always occur. A fused character box remains unsegmented and a single character is wrongly segmented into more than one character box. In either case, the post-processing phase must ensure that the word corresponding to the correct segmentation, is always included in the words which are searched. To ensure this, we form words with the IOCR output corresponding to every alternative segmentation.

Coming to classification errors, we note that a ranking error pushes the true character to a lower position. Therefore, we form multiple aliases of the word from the output of IOCR using top 3 choices for each character. Here is an example:
input word: हमारा
IOCR output: (ह, द) (म, भ) (ा) (र, द) (ा)
words formed: हमारा दमारा हभारा दभारा हमादा दमादा हभादा दभादा
The true word is one of the words formed if every character of the word is in top 3 choices. Number of aliases is at most 9 for *short* words and 27 for three character words and so on. The number of alias words goes up substantially if top 3 choices for modifiers are also used for forming the alias words. To restrict number of alias

words, we use the following two measures:

1. If the confidence figure associated with a character is above a certain threshold, second and third choices are not used for forming the alias words. The threshold is empirically decided.

2. Further search is not done if the closest match is within a preset threshold distance. The distance is the cumulative penalty for the substitutions required to map the IOCR word to dictionary word. The distance between a dictionary word and the IOCR word is calculated by assigning a penalty of '0' for a match between the two characters. The distance calculation table is shown in figure 49. During the testing phase of our text reading system, we collected the character classes which the IOCR tends to confuse. These confusions are collected and provided to the matching process. The penalty of a substitution is '1' when the substituted character is a known confusion of the character being substituted and the confidence figure associated is low. If the confidence figure is high, the penalty is increased to '3'. When the substitution is not a known confusion and only the bar property is same, the penalty figure goes up further to '4'. The penalty figure becomes '6' if the bar property is also violated. A substitution for the core modifier ' ि' incurs a penalty of '10' because the recognition rate for this modifier is very high. Figure 50 shows the confusion classes for our system. A lower modifier is ignored if the two words do not have a lower modifier at matching positions. This incurs a penalty of '2'. Devanagari script has a lower modifier that looks like a '.'(dot). Sometimes, noise is detected as a dot and sometimes a dot is missed. This rule takes care of both the situations. The difference in the length is multiplied by six and added to the distance.

The algorithm for the search strategy is given in figure 51. The word is checked in the selected short word partition if the word contains two or less core characters. The search ends if an exact match is found. If the word is not short, the word envelop and tags of the IOCR word are used to select partitions for search. While extracting the tags from the IOCR word, we also note the position of both the characters of the tag in the IOCR word and the number of characters between them. In other words, the two character tag becomes $n_1 ch_1 n_2 ch_2 n_3$ where n_1 and n_3 are the positions

situation	penalty assigned
1. The difference in length is d	6 * d
2. The character being mapped has low confidence figure and the character being substituted is a known confusion	1
3. The character being mapped has high confidence figure and the character being substituted is a known confusion	3
4. The vertical bar property of the character being substituted and the substitution matches	4
5. The vertical bar property of the character being substituted and the substitution does not match	6
6. The core modifier ı is being substituted	10
7. A top modifier is being substituted	2
8. A lower modifier is being substituted	2
9. A half character or an upper or lower modifier is ignored	2
10. all other substitution	10

Figure 49: Distance calculation rules used by matching process.

of ch_1 with respect to the beginning and end of the word respectively; n_2 is the number of characters between ch_1 and ch_2. While matching a word of the selected partition with IOCR word, we use only those words whose length is at most one less or more than the IOCR word. After checking the length, we then check the relative positions of the tag characters. If the relative positions are off by at most one, further matching is done. After exhausting all words of the selected partition, the next tag is used to select another partition. Whenever an exact match is found or the distance is less than the preset threshold, the search is terminated. The effect of the threshold on the performance, number of partitions probed and words compared are described in the next section.

The algorithm for comparing characters of the IOCR word and the dictionary word is given in figure 52. Initially, the characters at the beginning positions of words are compared. If the penalty is less than one, the positions of characters to be compared next is advanced by one for the IOCR word and the dictionary word. If the penalty is more than one and if the character of dictionary word is a half form character and the character of the IOCR word is not a half character, we advance the position

Character Confusion Matrix for the output of IOCR			
IOCR output	Possible True Chars.	IOCR output	Possible True Chars.
अ	ख क्ष भ	इ	ह ड
उ	र	ऊ	रु
र	द	घ	अ ध
ज	न व ब	म	य प
ट	द ड	य	प म
त	ल	व	ब
ल	त	थ	य
न	व ब	प	य म भ
भ	म प		

Figure 50: Character Confusion Matrix for the output of IOCR.

of the character to be compared to the next character only for dictionary word. If we find a lower or upper modifier in the IOCR word and no corresponding modifier in the dictionary word, we advance the IOCR character position by one. In each case, a penalty is added to the distance according to Figure 49. The difference in the length is checked at the end and the penalty is added to the distance.

6.6 Experimentation and Discussion

Let us consider the true input which is shown in figure 53(a) as an example. The output of the IOCR after composition is shown in figure 53(b). The word composed from highest confidence symbol sequence is shown here. The verification process searches for an exact match in the dictionary. Closest match is selected from the dictionary if an exact match is not found which is shown in figure 53(c).

The system has been tested on nine printed documents. We tested the system with the threshold of 3.0 and 1.0 for terminating the search. The results are presented in tables 10, 11, 12, 13, 14 and 15 for font I. The results are presented in tables 16, 17, and 18. The number of words formed per true word for each document is

```
initialize top_three_words;
initialize top_three_dis;
for every alias word formed from the output of IOCR for the true word
  if ( no. of core characters =< 2 )
  {   use no. of characters, lower, top and core modifiers to
       select short word partition;
      search selected short word partition retaining
        top three choices in local_top_three based on the distance;
      if ( the distance of top choice is zero )
         exit; /* from for loop */
      else
      { store top three choices from top_three_choices and
              local_top_three into top_three_choices;
         store the corresponding distances in top_three_dis;
         repeat the search process for next alias;
      }
   }
   else
   {   use no. of characters, lower, top and core modifiers to
        select long word partition;
      extract tags from IOCR word;
      for each tag {
         search selected tag word partition retaining
             top three choices in local_top_three based on the distance;
         if ( the distance of top choice is zero or
            less than a preset threshold)
           exit; /* from for loop of tags  and for words */
         else
         { store top three choices from top_three_choices and
               local_top_three into top_three_choices;
           store the corresponding distances in top_three_dis;
           repeat the search process for next tag;
        - }
       }
       repeat the search process for next alias;
    }
  }
  return top_choice_choices and top_three_dis;
```

Figure 51: Algorithm for partitioned dictionary search.

```
in_pointer = 0;
dic_pointer = 0;
while ( end of dic_word or end of iocr_word is not reached )
{
  dis= chmatch(iocr_word[in_pointer],dic_word[dic_pointer]);
  if ( dis < 1 )
  {
      acc_dis = acc_dis + dis;
      inc in_pointer;
      ine dic_pointer;
  }
  else if (!(ishalf(iocr_word[in_pointer])) &
             ishalf(dic_word[dicp_ointer])
  {
      acc_dis = acc_dis + dis;
      inc in_pointer;
  }
  else if(lower_modifier(iocr_word[in_pointer])) &
              !(lower_modifier(dic_word[dicp_ointer])
  {
      acc_dis = acc_dis + dis;
      inc in_pointer;
  }
  else if(upper_modifier(iocr_word[in_pointer])) &
               !(upper_modifier(dic_word[dicp_ointer])
  {
      acc_dis = acc_dis + dis;
      inc in_pointer;
  }
  else
  {
      acc_dis = acc_dis + dis;
      inc in_pointer;
      ine dic_pointer;
  }

  acc_dis = acc_dis + absolute ( in_pointer - dic_pointer );
}
```

Figure 52: Algorithm for comparing two words and calculating the distance.

(a) True input words

वर्ष दौरान भारतीय अर्थव्यवस्था भुगतान संतुलन

(b) Composed words (using highest confidence characters)

वपै दौरान भारलीय अर्थव्यवस्या प ् ा ा तान संतुलन

(c) Output of the verification process

वर्ष दौरान भारतीय अर्थव्यवस्था भुगतान संतुलन

Figure 53: Sample output from verification process.

given in first column of these tables. The number of partitions and words which are probed are also given in the same table. When an exact match is found, number of partitions and words searched are less compared to those searched when an exact match is not found. Words that contain reject errors or substitution errors cause the selected partition to be searched exhaustively. We have counted these numbers for *short* words, 3 character words and the remaining words separately. The recognition performance at word level before and after post-processing is presented in the next chapter. Among all the words, it is most difficult to provide correct alternative in case of *short* words. The alternatives are usually too many with same distance. For instance, for the IOCR word आना, the alternatives are the following:
आना, खाना, जाना, ठाना, ताना, थाना, दाना, नाना, पाना, बाना, भाना, माना, लाना, साना.
All of the above words are valid dictionary words and have same distance from the true word. In other words, if first letter is not the true character in आना, it cannot be corrected. This is the reason, we have not used any threshold for stopping the search in case of short words. Sometimes a ranking or substitution error maps the true word to another dictionary word. These errors go undetected at the word level.

A question we have not addressed so far is: should a non-dictionary word be always mapped to a dictionary word? The answer is no, and to avoid unnecessary mappings,

we have made the penalty of substitution higher for the characters which have a high confidence of figure associated. In addition, the mapping is not accepted if the minimum distance is more than a preset threshold.

	total words	Average no. of words formed/true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	133	5.24	1.59	169	33	2702
Doc II	140	8.87	2.90	519	18.06	2034
Doc III	93	10.41	5.21	823	26.48	1367
Doc IV	74	10.94	2.78	294	26.53	1527
Doc V	125	8.57	1.53	234	22.72	1804
Doc VI	144	23.21	4.49	681	11.0	1398
Doc VII	148	5.66	1.41	162	8.84	1000
Doc VIII	184	6.83	3.41	467	15.84	1189
Doc IX	149	6.36	4.74	655	13.97	1236
Overall	1190	85.64	28.06	4004	176.44	14257
Average		9.51	3.11	444.88	19.60	1584.11

Table 10: Dictionary Search Performance for *short* words when distance threshold is 3.0 for terminating the search (Font I).

6.7 Summary and Conclusions

We have presented a hierarchical partitioning scheme and search algorithm for the correction of optically read Devanagari characters of text recognition system for Devanagari script. The preference is given to those mappings which use known IOCR confusions. The methodology described here can be easily adapted system to spell checker by providing a new confusion matrix modeling the spelling error behaviour[72]. The new confusion matrix will be based on the knowledge of typographical errors and grammatical errors. With the speech recognition system,

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search: match is found: partitions probed/ true word	search: match is found: words matched/ true word	search: match is not found: partitions probed/ true word	search: match is not found: words matched/ true word
Doc I	133	11.93	3.80	376	47.58	2325
Doc II	140	13.67	8.63	619	39.07	2798
Doc III	93	17.13	13.70	1028	54.12	2357
Doc IV	74	20.46	5.42	381	57.90	2858
Doc V	125	16.39	22.34	601	44.52	2716
Doc VI	144	33.48	46.30	837	34.90	2342
Doc VII	148	12.72	2.80	389	40.58	1834
Doc VIII	184	12.03	10.44	742	35.95	1719
Doc IX	149	15.40	10.1	724	58.28	2469
Overall	1190	153.21	126.45	5316	355	21418
Average		17.02	14.05	590.66	39.44	2379

Table 11: Dictionary Search Performance for *short* words when distance threshold is 1.0 for terminating the search (Font I).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search: match is found: partitions probed/ true word	search: match is found: words matched/ true word	search: match is not found: partitions probed/ true word	search: match is not found: words matched/ true word
Doc I	45	4.57	1.64	6.6	11.7	182
Doc II	47	7.89	8.08	44.20	31.17	92.21
Doc III	26	9.84	11.91	26.91	33.14	222
Doc IV	22	4.31	5.35	12.24	16.87	44.62
Doc V	45	8.22	3.27	11.72	35.55	159
Doc VI	47	9.14	7.35	16.8	12.48	30.22
Doc VII	43	8.37	3.35	10.06	92.66	361.25
Doc VIII	57	6.94	10.25	25.25	37.92	113.24
Doc IX	61	2.80	1.76	7.51	46.66	261
Overall	393	62.08	52.96	161.29	318.15	1465.54
Average		6.98	5.88	17.92	35.35	162.83

Table 12: Dictionary Search Performance for three core character words when distance threshold is 3.0 for terminating the search (Font I).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	45	12.08	30.25	107.06	59.5	2451
Doc II	47	24.18	84.14	396.96	77.8	488.75
Doc III	26	20.07	22.26	61.13	105.92	623
Doc IV	22	17.58	10.73	46.20	139	784
Doc V	45	15.69	33.65	167.46	69.80	315.35
Doc VI	47	28.56	43.96	287.16	122	728.35
Doc VII	43	15.39	14.51	33.64	131	439.16
Doc VIII	57	19.13	36.59	111.90	120	341
Doc IX	61	10.03	24.17	128	62.66	387.33
Overall	393	162.71	300.26	1339.01	887.69	6557.94
Average		18.07	33.36	148.77	98.63	728.66

Table 13: Dictionary Search Performance for 3 core character words when distance threshold is 1.0 for terminating the search (Font I).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	38	1.71	5.72	8.32	23.62	40.25
Doc II	15	3.86	4.57	8.85	17.25	40
Doc III	32	3.125	5.0	10.53	19.23	17.70
Doc IV	32	12.21	3.85	9.42	7.45	22.18
Doc V	33	6.0	2.04	7	34.5	108
Doc VI	20	10	45	9.57	108	49
Doc VII	55	3.6	2.54	11.87	54.71	274.85
Doc VIII	47	2.48	4.34	9.17	13.61	20.27
Doc IX	34	1.58	3.73	7.56	16.72	34.81
Overall	306	44.56	76.79	82.29	295.09	607.06
Average		4.95	8.53	9.14	32.78	67.45

Table 14: Dictionary Search Performance for four or more core character words when distance threshold is 3.0 for terminating the search (Font I).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	38	5.17	6.25	11.14	45	68.46
Doc II	15	8.45	12.36	18.27	44.88	127.33
Doc III	32	9.25	55.17	276	24.8	42.2
Doc IV	32	21.56	14.11	37	32.4	76.2
Doc V	33	11.09	22.05	57.36	48.47	105.57
Doc VI	20	17.05	95	65	131	73.72
Doc VII	55	5.92	6.87	23.85	168.70	328.83
Doc VIII	47	6.91	15.94	52.5	50.25	214.33
Doc IX	34	3.82	6.73	11	51.81	154
Overall	306	89.22	234.47	552.12	497.31	1036.64
Average		9.91	26.05	61.34	55.25	115.18

Table 15: Dictionary Search Performance for four or more core character words when distance threshold is 1.0 for terminating the search (Font I).

confusion matrix will come from the knowledge of phonemes which get confused with other phonemes. The partitioning scheme will have to be phoneme based.

	total words	Average no. of words formed/true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	232	11.84	4.03	394	51.13	2800
Doc II	291	19.34	9.36	608	34.23	2136
Doc III	267	14.31	24.56	721	38.56	1726
Doc IV	304	16.49	36.38	821	53.48	1938
Doc V	286	17.38	11.56	456	59.46	2432
Doc VI	114	14.59	12.48	521	37.46	1626
Doc VII	262	12.36	5.09	317	48.32	1728
Doc VIII	141	15.39	17.21	418	64.46	1622
Doc IX	52	21.65	11.38	609	60.06	1721
Doc X	130	17.49	12.42	411	35.32	2712
Overall	2079	160.84	144.47	5276	482.48	20432
Average		16.08	14.44	527.60	48.24	2043.2

Table 16: Dictionary Search Performance for *short* words when distance threshold is 1.0 for terminating the search (Font II).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	117	30.90	76.45	321.26	136	810.45
Doc II	93	10.45	26.38	92.26	60.36	315
Doc III	80	20.46	38.84	218.54	70.36	222
Doc IV	108	17.32	41.54	170.46	96.38	654
Doc V	72	18.07	33.36	111.90	105.92	623
Doc VI	52	19.34	40.38	154.90	124	738
Doc VII	98	24.56	84.31	152	76.36	1608
Doc VIII	64	12.98	24.98	94	36.91	1403
Doc IX	24	15.67	24.56	304	46.36	1364
Doc X	63	21.34	36.34	198	54.30	801
Overall	771	191.09	427.24	1817.32	806.95	9538.45
Average		19.10	42.72	181.73	80.69	953.84

Table 17: Dictionary Search Performance for 3 core character words when distance threshold is 1.0 for terminating the search (Font II).

	total words	Average no. of words formed/ true word with consideration of touching/fragmented characters	search			
			match is found		match is not found	
			partitions probed/ true word	words matched/ true word	partitions probed/ true word	words matched/ true word
Doc I	33	14.17	21.21	57.38	60	169
Doc II	42	21.16	32.16	40	46	112
Doc III	36	6.22	11.32	55	78	206
Doc IV	52	7.68	12.84	18	170	514
Doc V	23	9.36	15.65	37	62	181
Doc VI	25	11.58	17.38	55	94	274
Doc VII	40	9.34	12.34	46	372	1005
Doc VIII	16	11.68	14.34	48	305	630
Doc IX	9	16.42	26.24	154	296	918
Doc X	17	14.34	22.48	91	137	524
Overall	293	121.95	185.96	601.38	1620	4533
Average		12.19	18.59	60.13	162	453.3

Table 18: Dictionary Search Performance for four or more core character words when distance threshold is 1.0 for terminating the search (Font II).

Chapter 7

Devanagari Text Recognition: Implementation and Experimentation

In earlier chapters, we have talked about various modules of our document reading system. In this chapter, all the modules have been put together and a comprehensive view of the system is provided. The system consists of a training phase which precedes the recognition phase. The training phase acquires prototypes for the character set for Devanagari script for various fonts and styles. The training phase has been described in this chapter. Some of the sample runs are also included in this chapter. The overall system performance on various text documents is also given.

7.1 The Document Reading System

The DFD (Data Flow Diagram) for the document reading system is shown in figure 54. The control flow diagram is shown in figure 55.

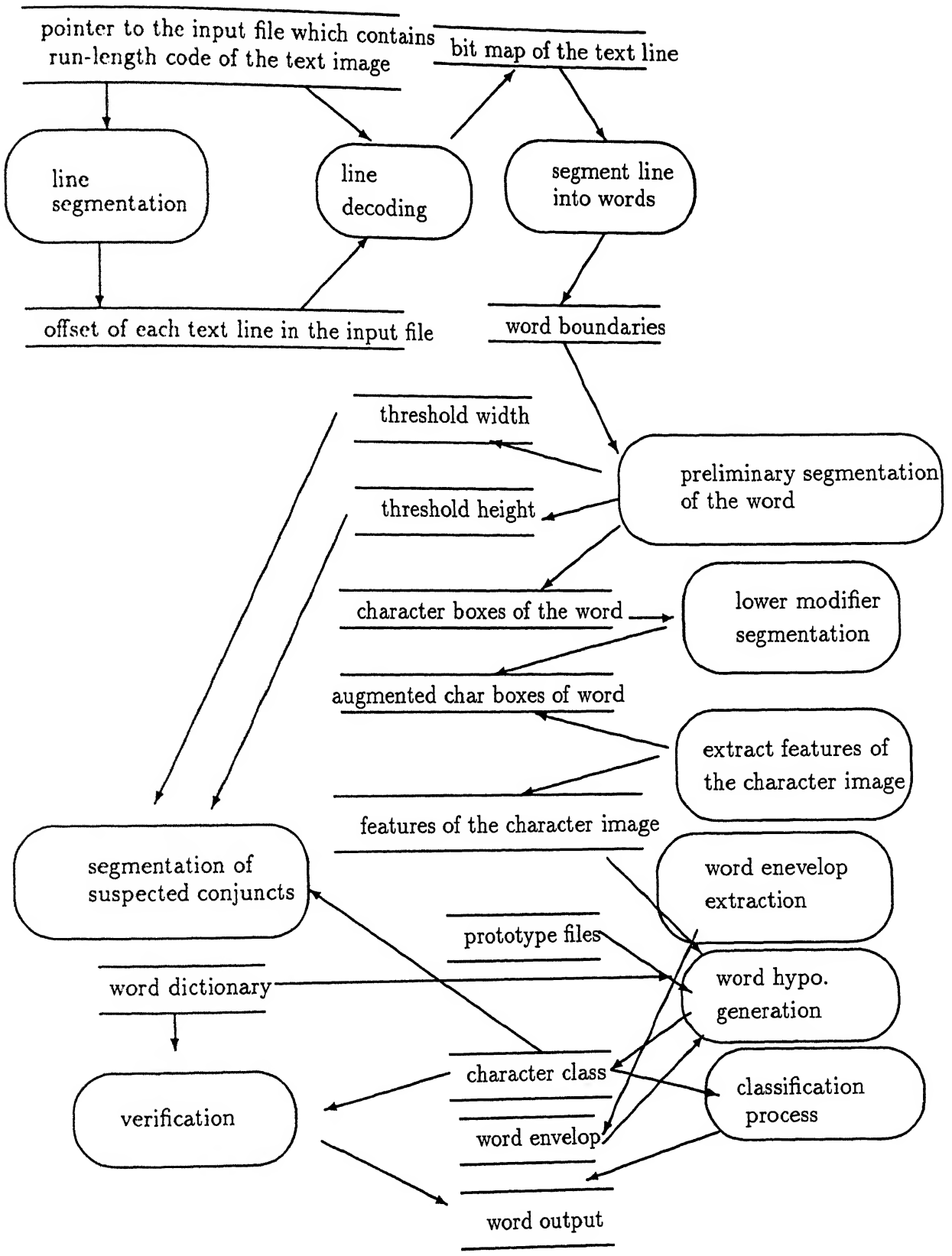


Figure 54: Data Flow Diagram of the Document Reading System.

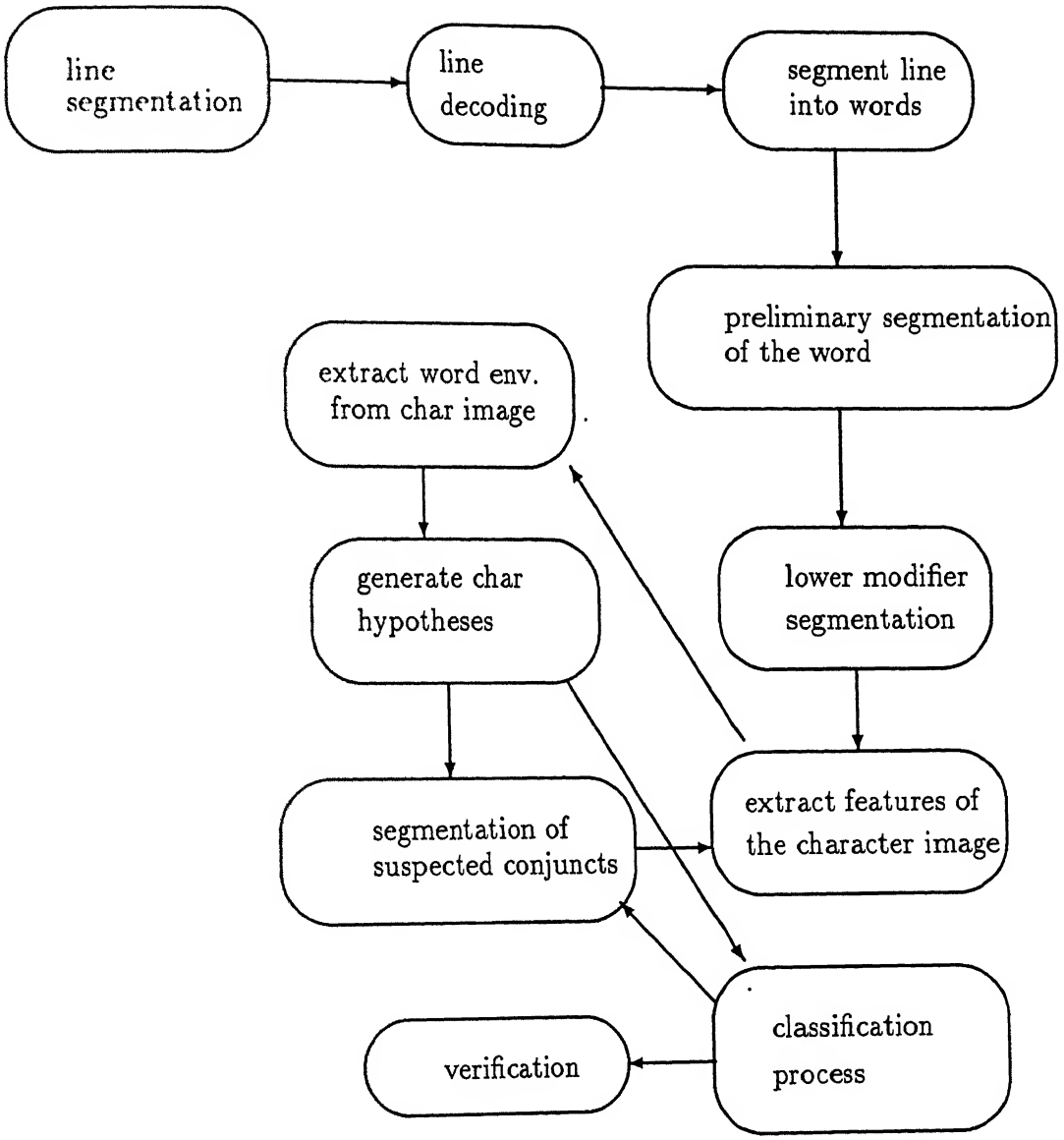


Figure 55: Control Flow Diagram of the Implemented Document Reading System.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1.	1	0	number of pixels in the scan line													
2.	1	1	number of pixels in the scan line													
3.	0	0	number of nibbles representing run length data													
4.	0	1	number of nibbles representing run length data													

Table 19: Four different headers used in run length codes.

We assume that a preprocessing stage extracts the uniform text zones from the document image. Also, the tilt correction has been done and portrait printing mode has been used. The size of the paper is A4 (11 inches x 8 inches). The image is scanned at 200 DPI in monochrome mode. We used a CANON IX-12 document scanner at 200 DPI resolution. Our experience shows that at lower resolution, breaks in symbols get introduced and at higher resolution, symbols tend to merge with each other due to the distortion introduced by digitization process. For most of the printed text, 200 DPI suits well. The bit map of the document requires approximately 500 KBytes memory for a 11 * 8 inch page scanned at 200 DPI. This requirement has been reduced by a factor of 10 by using run length coding to store the image. The four headers consisting of two bytes is used for every pixel line is shown in table 19.

When all pixels of the scan line are zeros or ones, header 1 and 2 are used respectively. When the scan line consists of both zeros and ones, header 3 and 4 are used. Header 3 is used when first pixel is zero and header 4 is used when first pixel is one. In case of headers 3 and 4, the following nibbles are read to get the information of the pixels of the scan line. We next explain each processing block of the figure 54. The implementation details of each of the processing block of figure 11 are presented in the following subsection. These processing blocks correspond to the tools used in the Blackboard model as outlined in figure 11 in chapter 3.

7.1.1 Line Identification

This process is the first process of figure 11 of chapter 3. This process consists of three subprocesses, two for text line segmentation and the other for converting the run length coded image into corresponding bit-map.

- *Line Segmentation*

The image file is read, one pixel line at a time and the offsets (beginning and end) for each pixel line is stored in an array referred to as *offSets[MaxPixelLines][2]*. Along with this, the horizontal histogram of the image is also made which is used for the segmentation of the text into text lines. We have not implemented the preprocessing stage. Therefore, an estimate for the line height which the text zone extraction module would have provided is not available. The line segmentation consists of two passes as explained in section 4.1 of chapter 4 which constitute *phase I and phase II* of this process. After lines have been segmented, the *lineOffsets[MaxTextLines][2]* for each text line is saved. This enables the succeeding modules to read the required text line form the run-length coded file without converting it into corresponding bit-map. This module displays the total number of lines in the text.

- *Line Decoding*

This process is not shown in figure 11 of chapter 3, because it is required only for getting the bit-map of the text line. If the bit-map of the document is the input to the recognition system, this process will become redundant. The user specifys the range of the text line for processing. The decoder converts the lines, one at a time, into bit-map and stores it into an array. The image is stored in a two-dimensional array of unsigned char *TxtLineImg[100][200]* which can store a text line with 1/2 inch height and 8 inch breadth at 200DPI.

Word Segmentation/isolation The process is same as in figure 11 of chapter 3. The word segmentation/isolation module takes the bit map of a text line and makes a vertical histogram of the image. Every vertical gap corresponds to the word boundary. The details of this algorithm have been explained in section 4.2 of chapter 4. For every segmented word, the position of the header line is also located with the help of the horizontal histogram of the word. This information is stored with every word. The data structure for storing the above information is shown in the figure 56. The items on lines 4, 7 and 8 are explained in the next few paragraphs of this section.

```
typedef struct word {
 1.  int        txtLineNum;            /* text line number */
 2.  int        wordNum;               /* word num in the current line */
 3.  int        lMatra;                /* set if lower matra present */
 4.  CharInfo   cboxes[MAX\_SYMUNITS]; /* constituent chars */
 5.  int        numChars;              /* num of char boxes */
 6.  int        hLinePos;              /* position of the header line */
 7.  char       words[MAXWORDS][2 * MAX\_SYMUNITS];
                                       /* output of the OCR */
 8.  int        charPos[MAXWORDS][MAX\_SYMUNITS];
                                       /* position of the constituent chars */
} WordInfo;
```

Figure 56: Data structure for storing the Word Information.

Preliminary Segmentation of the Word: This process is *Symbol Extraction* of figure 11 of chapter 3. This is *Symbol Extraction: Phase I* of the figure 14 of chapter 3. The word is segmented into two regions; the region above the header line and the region below the header line. The characters and symbols of these two regions are obtained independently by the algorithm which has been explained in section 6.1 of chapter 4. The coordinates, the height and width information of the units obtained after the preliminary segmentation is stored in the *cboxes* of figure 56. This data structure is shown in figure 58. The *left* and *top* are made zero and the translation with respect to the text line is stored in *transTop* and *transLeft*. The

strip to which the character or the symbol belongs is stored in the field *tcb*. The *posNxt* tells the position of the next unit in the cboxes array. The following example will make the use of *posNxt* clear.
The true word is राष्ट्रीय. The image boxes of this word, their position and *posNxt* is the following:

character	position in cboxes array	posNxt
र	1	2
ा	2	3
ष्ट्र	3	4
ी	4	6
य	5	-1
ं	6	5

Figure 57: The character boxes, their position in the cboxes array and field posNxt after preliminary segmentation.

The use of *posNxt* enables us to segment the top strip and the remaining image independently and then set the *posNxt*. The *posNxt* is set by the first phase of the composition module which has been explained in the section 6.1 of chapter 6. Item on lines 10, 11, 12, 13 and 14 of figure 58 are explained in the next few paragraphs of this section. The height and width information of the core characters obtained after preliminary segmentation is used for setting the threshold height and threshold width referred to as *thHeight* and *thWidth* respectively. This has already been explained in chapter 4. The association of top modifiers is also done at this stage and accordingly the *posNxt* is set. Value -1 of *posNxt* indicates the end of the character array.

Segmentation of Lower Modifiers: This process is *Symbol Extraction* of figure 11 of chapter 3. This is *Symbol Extraction: Phase III* of the figure 15 of chapter 3. The image boxes of core strip which are taller than the *thHeight* are further segmented if possible. The segmentation algorithm has already been discussed in the section 4.5

```
typedef struct cordImgBox {
 1.  unsigned char cImage[100][CharWidth];
 2.  int      left;
 3.  int      right;
 4.  int      top;
 5.  int      bottom;     /* coordinates */

 6.  int      transTop;  /* translation with respect */
 7.  int      transLeft; /* to the text line */

 8.  char     tcb;       /* strip T(op), C(ore), B(ottom) */
 9.  int      posNxt;    /* next box  */

10.  int      half;      /* half character */
11.  int      posAlt;    /* alternates   boxing          */

12.  int      numAlts;    /* number of output of OCR */
13.  char     alts[MAX\_ALTS][3]; /* each output */
14.  float    conf[MAX\_ALTS];    /* confidence level */
} CharInfo;
```

Figure 58: Data structure for storing the Character/Symbol Information.

in chapter 4. The image boxes are added to the cboxes array and the *posNxt* and *posAlt* fields of the image box segmented are modified. The modified array cboxes, the *posNxt* and *posAlt* of the sample word shown in figure 57 are shown in figure 59.

Often times, the lower modifiers cover the vertical gap between two adjacent characters. Therefore, the preliminary segmentation phase is unable to extract such characters. After removing the lower modifier, the core image is again passed through the preliminary segmentation phase.

The lower modifier segmentation process is invoked after the *thHeight* becomes available. We do not wait for the output of the classification process as we do in case of the wide character boxes. The association of lower modifiers with core

character	position in cboxes array	posNxt	posAlt
र	1	2	-1
ा	2	3	-1
ष्ट्र	3	4	7
ा	4	6	-1
य	5	-1	-1
◌	6	5	-1
ष्ट	7	8	-1
◌	8	4	-1

Figure 59: The character boxes, their position in the cboxes array and, fields posNxt and posAlt after lower modifier segmentation.

characters is done immediately after segmenting the lower modifier.

Classification Process

This process is *Symbol Recognition* of figure 11 of chapter 3. The prototypes which are extracted during the training phase are read into the memory when the system is started. These prototypes are used by the classification process. There are three classification processes, one for each of the following:

1. Top modifiers; This is *Symbol Recognition: Phase I* of the figure 15 of chapter 3.
2. Lower modifiers. This is *Symbol Recognition: Phase II* of the figure 15 of chapter 3.
3. Core characters, This is *Symbol Recognition: Phase III* of the figure 15 of chapter 3.

The data flow diagram for each of these processes have been shown in figures 60 and 61. The candidate sets for FULL BOX, UPPER HALF and LOWER HALF boxes have already been explained in section 2.6.1 of chapter 2. The character classes based on vertical bar feature and number of junctions formed with the header line have also been included in the same section. We obtained 41 classes based on

horizontal zero crossing vector. There is some overlap between classes as is clear om the table 20. The output of the classification process is stored in the *alts* and

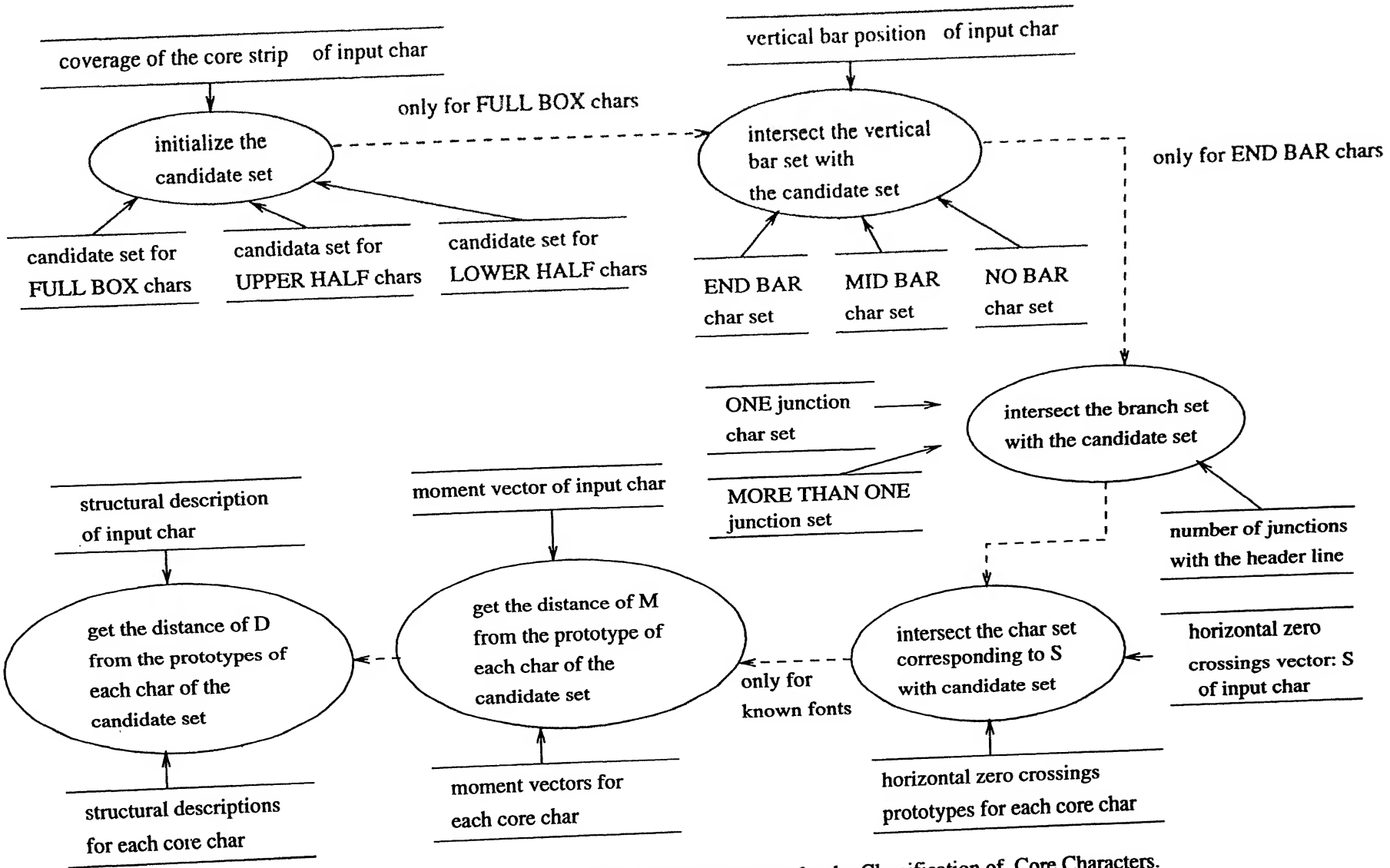
number of true character classes	55
number of classes based on S	41
average number of characters in a class	6
minimum number of characters in a class	1
maximum number of characters in a class	22

Table 20: Statistics of Classes based on Modified Horizontal Zero Crossing Vector: S

onf arrays of the data structure shown in figure 58. If the character is classified to a ngle known class, the *numAlts* of the figure 58 is set to 1 and the character class is ored in the fields *alts[0]*. The associated confidence is stored in *conf[0]*. If there are ore than one classes, they are stored in the decreasing order of confidence in the rray *alts* and the confidence is stored in *conf* array. The field *numAlts* is set to the otal number of classes. The maximum number of classes which has been observed uring experimentation is six. The ranking of the true character never falls below ne sixth place. The unknown class is represented by asterisk (*). The confidence vel has been observed to vary between 5.08 to 3.0 for the true character. If the onfidence level goes below 3.0, the classification is suspected to be wrong.

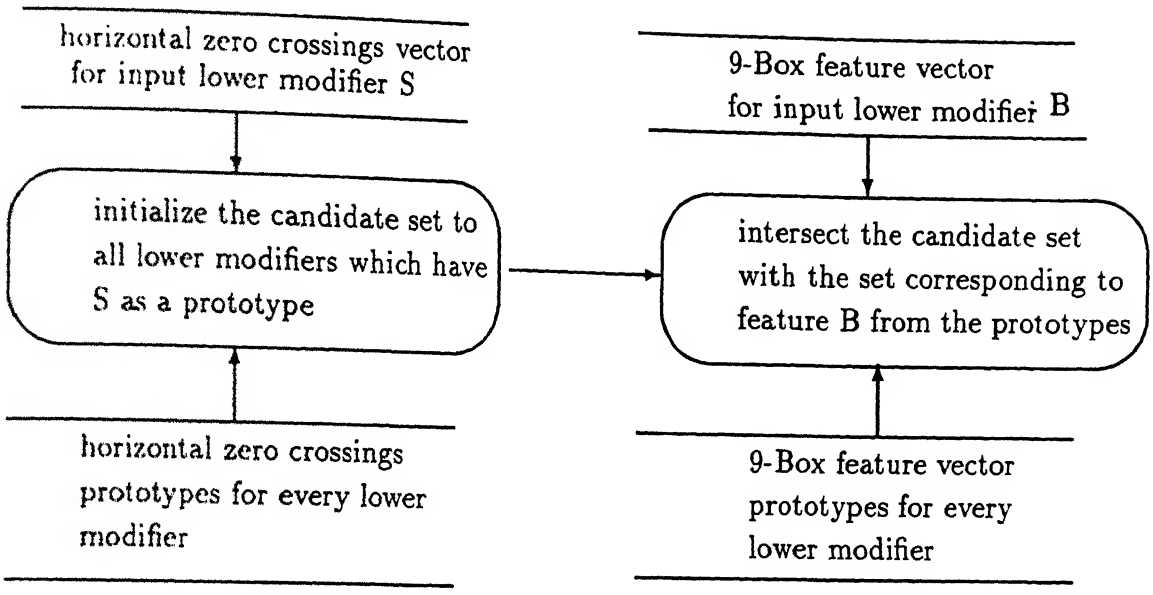
Hypotheses Generation Process The number of character boxes in each strip nd the vertical bar property of the middle (core) strip character boxes are used o generate word envelop information. The word envelop information is used to elect candidate words from a word dictionary which has been partitioned using vord envelop in a hierarchical way. The hypothesis set for each character box is onstituted by corresponding character of each selected word.

Segmentation of the Conjuncts This process is *Symbol Extraction* of figure 11 of chapter 3. This is *Symbol Extraction: Phase II* of the figure 15 of chapter 3. f the confidence level of a wide character is below 3.0 or the candidate character et becomes empty, further segmentation of the character box is attempted. This lgorithm has already been explained in the chapter 4. If the symbol is extracted



Fogure 60: Data Flow Diagram including the Control Flow for the Classification of Core Characters.

(a) Data Flow Diagram for the Lower Modifiers:



(b) Data Flow Diagram for the Upper Modifiers:

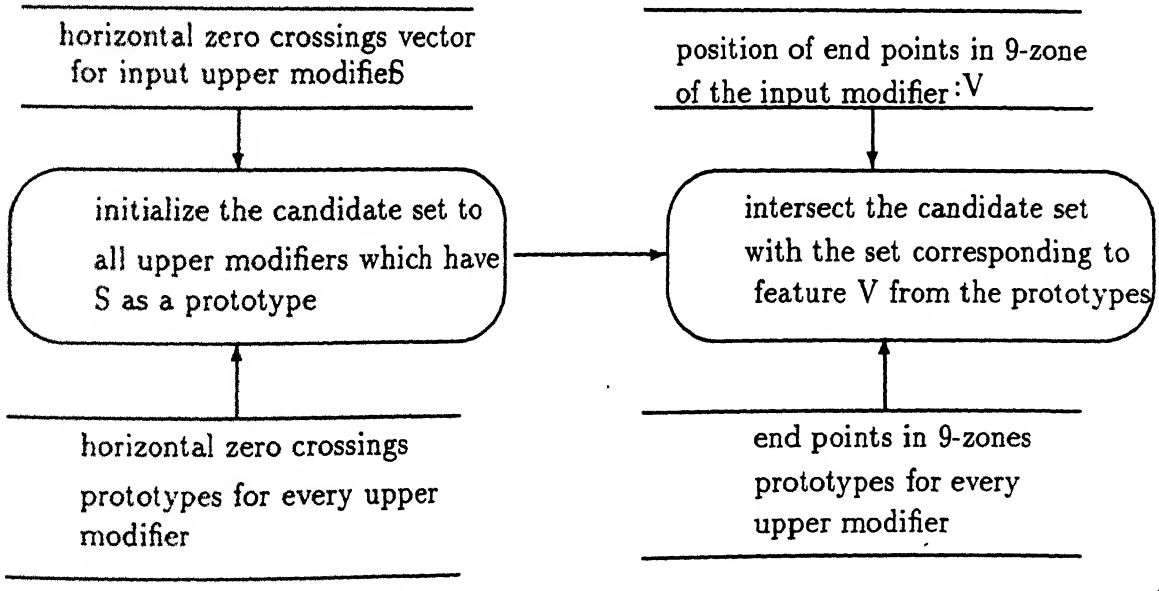


Figure 61: Data Flow Diagram including the Control Flow for the Classification of Lower and Upper Modifiers.

from the left part of a conjunct, the half flag is set to indicate the nature of the character to be the derived half form. This information is used by the classification process. The newly obtained character boxes are added to the cboxes array and the fields of the old image box are modified. The modified array *cboxes*, the *posNxt* and *posAlt* of the sample word shown in figure 59 are shown in figure 62.

character	position in cboxes array	posNxt	posAlt
र	1	2	-1
ा	2	3	-1
ष्ट्र	3	4	7
ा	4	6	-1
य	5	-1	-1
ं	6	5	-1
ष्ट	7	8	9
ं	8	4	-1
६	9	10	-1
ट	10	4	-1

Figure 62 The character boxes, their position in the cboxes array and, fields posNxt and posAlt after conjuncts have been segmented.

The classification process is invoked on the newly created character boxes.

Word Composition This process is *Word Composition* of figure 11 of chapter 3. For this experimentation, we did not use the complete composition phase. The composition processor has been tested and the results have been reported in [71]. Moreover, we wanted to see the performance of the classification process and the verification process. In this implementation, only the sequences of the output of classification process are placed in the *word* field of the data structure shown in figure 56. The position of the characters which constitute the character sequence is put in the *charPos* array of the same data structure. The word sequences are formed in the order of decreasing confidence. Highest confidence word is formed first and verified.

Verification Process This process is *Word Hypothesis Generator and Verifier* of

figure 11 of chapter 3. Since the word sequences are not composed, we decompose the dictionary words while performing the matching. The highest confidence sequence is verified first. If an exact match is found, no more word sequences are tried. The matching algorithm has already been explained in the chapter 6. The confusions which the classification process made were provided to the verification process.

This completes the description of the document reading system. The output of the verification process is kept in a separate flat file. A sample input document page is shown in figure 63. The first five lines after preliminary segmentation and lower modifier separation are shown in figures 64 and 65. The rest of the lines are processed in the same manner. However, only first five lines are shown to illustrate the process. One may notice that the conjuncts and shadow characters remain unsegmented at this stage. The output of the classification process on the image of figure 65 is given in figure 66. For sake of clarity, the upper and lower modifiers have been omitted. Only the top choice of IOCR is shown in this figure. The numerals and ligatures have not been incorporated in this work. The numerals get segmented in an unpredictable manner due to the assumed presence of a header line. The lines 1-5 of the document after conjunct segmentation is shown in figures 67. The output of the classification process on the image of figure 66 is given in figure 68 and output after verification process is shown in figure 69.

7.2 Automated Trainer for Construction of Prototypes: A knowledge Acquiring Phase

To a reader who is not familiar with Devanagari script, the task of establishing correspondences of symbols obtained through optical processing and those of true text file, may seem to be a trivial one. However, it is soon realized that the correspondence can be established when there is a unique way of obtaining ordered symbol string of the script. In case of Roman script, the symbols are just the individual characters as the script is a mere juxtaposition of the characters. On

The document image

भारत में गरीबी ग्रामीण क्षेत्र में ही विद्यमान है ऐसी धारणा चिन्तन के स्तर पर मानी जाती रही है। इसीलिए गरीबी उन्मूलन के अधिकांश कार्यक्रम ग्रामोन्मुख रहे हैं। गरीबी की विकरालता तथा विशालता निस्सन्देह ग्रामीण क्षेत्रों में विराट रूप से परिलक्षित होती है, किन्तु शहरी क्षेत्र इससे अछूता हो, ऐसा नहीं है। शहरी क्षेत्रों में गरीबी का रूप भिन्न है। शहरी क्षेत्र भी जनसंख्या के अनुसार अलग-अलग श्रेणी में वर्गीकृत हैं और इन क्षेत्रों में गरीबी का विस्तार अलग-अलग है। महानगरों में जहां निम्नतम वर्ग के लोगों की संख्या अपेक्षाकृत अधिक होती है, वे तकनीकी रूप से गरीबों की श्रेणी में रखे जाने के पात्र नहीं होते क्योंकि उनकी आय का स्तर अपेक्षाकृत अधिक होगा। ऐसे महानगरों में मध्यम वर्ग का प्रतिशत भी अपेक्षाकृत ऊंचा है जिनकी आय 750/- रुपये प्रति माह से 2500/- रुपये प्रति माह होगी, लेकिन यह प्रतिशत छोटे नगरों या कस्बों में भिन्न होगा। वहां आय का स्तर अपेक्षाकृत कम होगा और वहां के निवासियों में गरीबी रेखा के नीचे रहने वालों का प्रतिशत ज्यादा होगा। लेकिन, यह वर्गीकरण गरीबी की भयावहता को कम नहीं कर पाता। वस्तुतः शहरी क्षेत्रों में जो लोग विपन्नता की स्थिति में जीवनयापन करते हैं, उनमें से अधिकांश गांवों से पलायन कर आये हुए लोग हैं जो आते ही रोजगार की तलाश में हैं। दिनोंदिन विस्तारित होती गंदी बस्तियों और फुटपाथों पर बसेरा करने वाले लोग और अन्य दीनहीन बेरोजगार वर्ग के लोग शहरी गरीबी के प्रत्यक्ष रूप हैं।

Figure 63: A Sample Document Page.

The image after preliminary segmentation (lines 1-5)

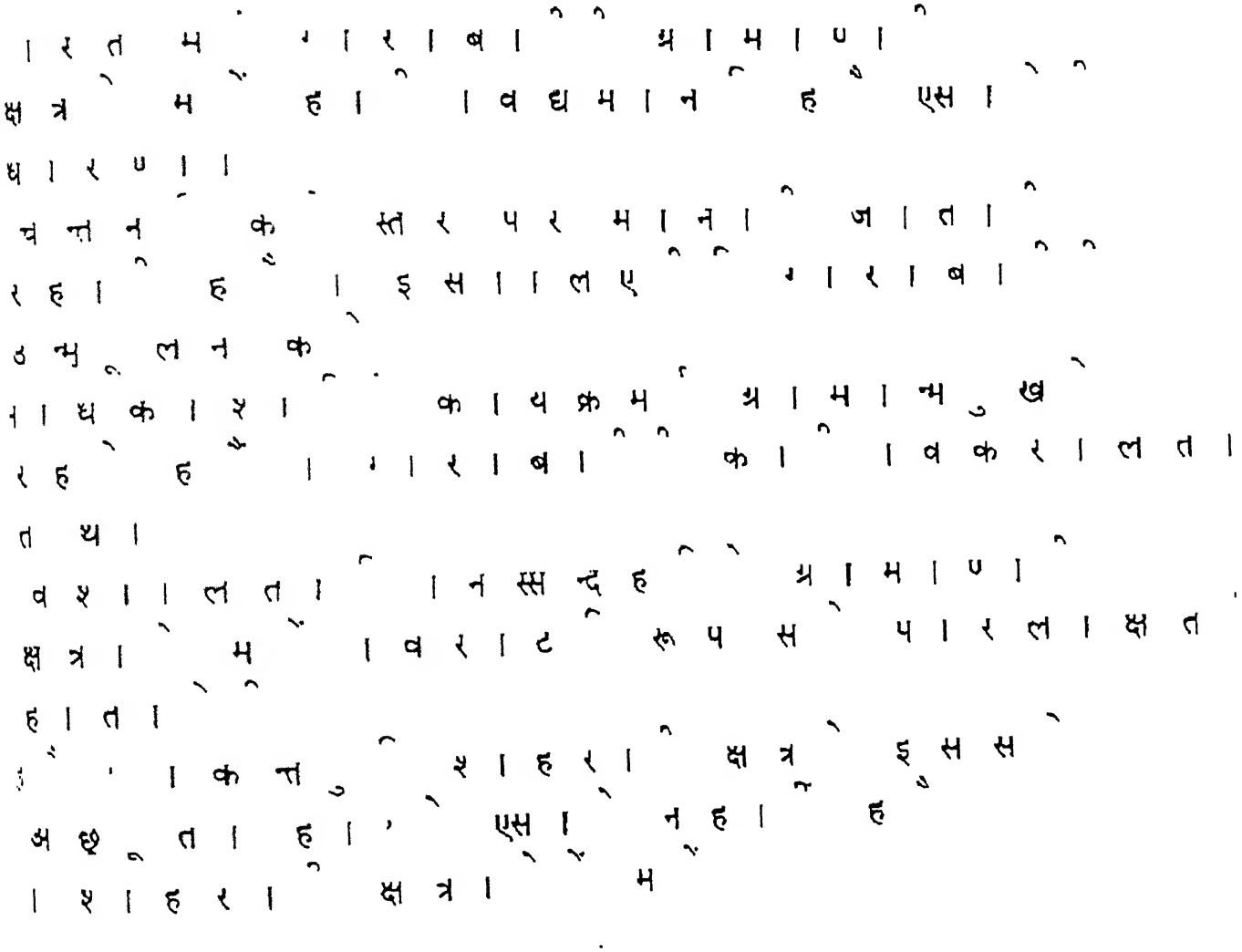


Figure 64: The image after preliminary segmentation (lines 1-5); lower modifiers and composite characters have not yet been segmented.

The image after lower modifier separation (lines 1-5)

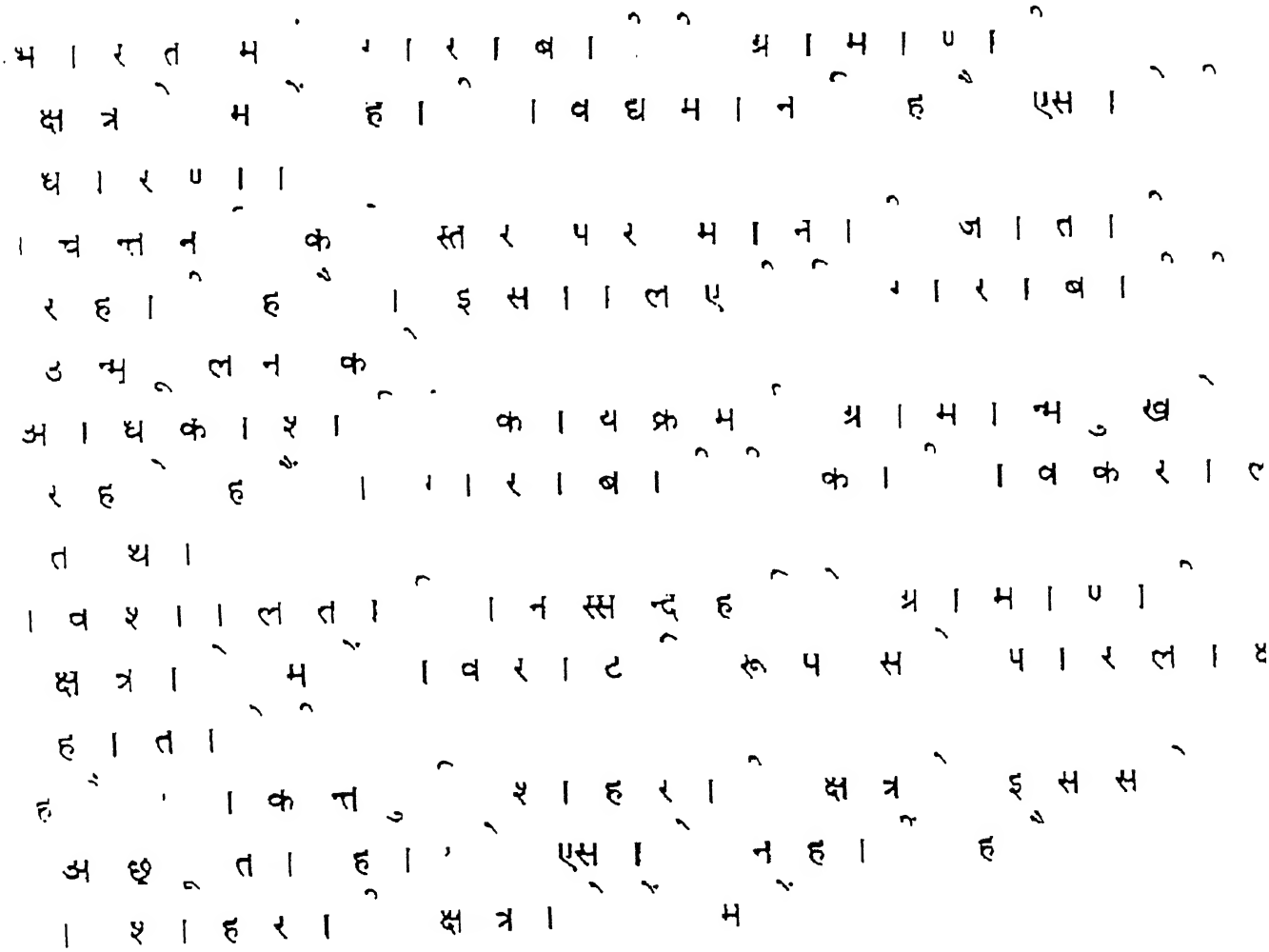


Figure 65: The image after lower modifier separation (lines 1-5); composite characters have not yet been segmented.

Output of IOCR

भारत में गरीबी यामीण क्षेत्र मे ही वियमान है ऐसी धारणा
चिसन के *र पर मानी जाती रही है। इसीलिए गरीबी उ*लन के
अधिकांश कार्यकम ग्रामो*ख रहे है। गरीबी की विकरालता तथा
विशालता नि**ह सामीण क्षेवो मे विराट रुप से परिलक्षित होती
है, किचु शहरी क्षेब इससे अछूता हो, ऐसा नही है। शहरी क्षेवों में
गरीबी का रूप थिव है। शहरी क्षेव थी वनसं* । के अनुसार
अलग-अलग वेणी में वर्गीकृत हैं और इन क्षेवों में गरीबी का
बिस्तार अलग-अलग है। महानगरों सें जहां नि*तम वर्ग के लोगों
की स* । अपेक्षाकृत यधिक होती है, वे तकनीकी रूप से गरीबों
की श्रेणी में रखे लाने के पाव नहीं होते क्पोंकि उनकी आय का *र
अ**कृत अभिक होगा। ऐसे महानगरों में म*म वर्ग का वतिशत
भी अपेक्षाकृत *चा है जिनकी आय *** रूपये प्रति माह से
**** रुपये प्रति माह होगी, लेकिन यह सतिशत छोटे नगरों या
क*ों में भिव होगा। वहां आय का *र अपेक्षाकृत कम होगा और
वहां के निबाक्षियों में गरीबी रेखा के नीचे रहने वालों का प्रतिशत
* । दा होगा। लेकिन, यह वर्गीकरण गरीबी की थयाबहता को कम
नहीं कर पाता। व*तः शहरी *त्रों में जो लोग विपवता की ।*ति
में जीवनयापन करते हैं, अनमें से अचिकांश गांबों से पलायन कर
आये हुए लोग हैं जो आते ही रोजगार की तलाश में हैं। दिनोंदिन
वि* रित होती गंदी ब।*यों और फुटपाथों पर बसेरा करने वाले
जोग और अ* दीवहीन बेरोजगार वर्ग के लोग शहरी गरीबी के
मलक्ष रूप हैं।

Figure 66: The output of the classification process; only the top choice is shown here; * represents a rejected character.

The image after conjunct segmentation (lines 1-5)

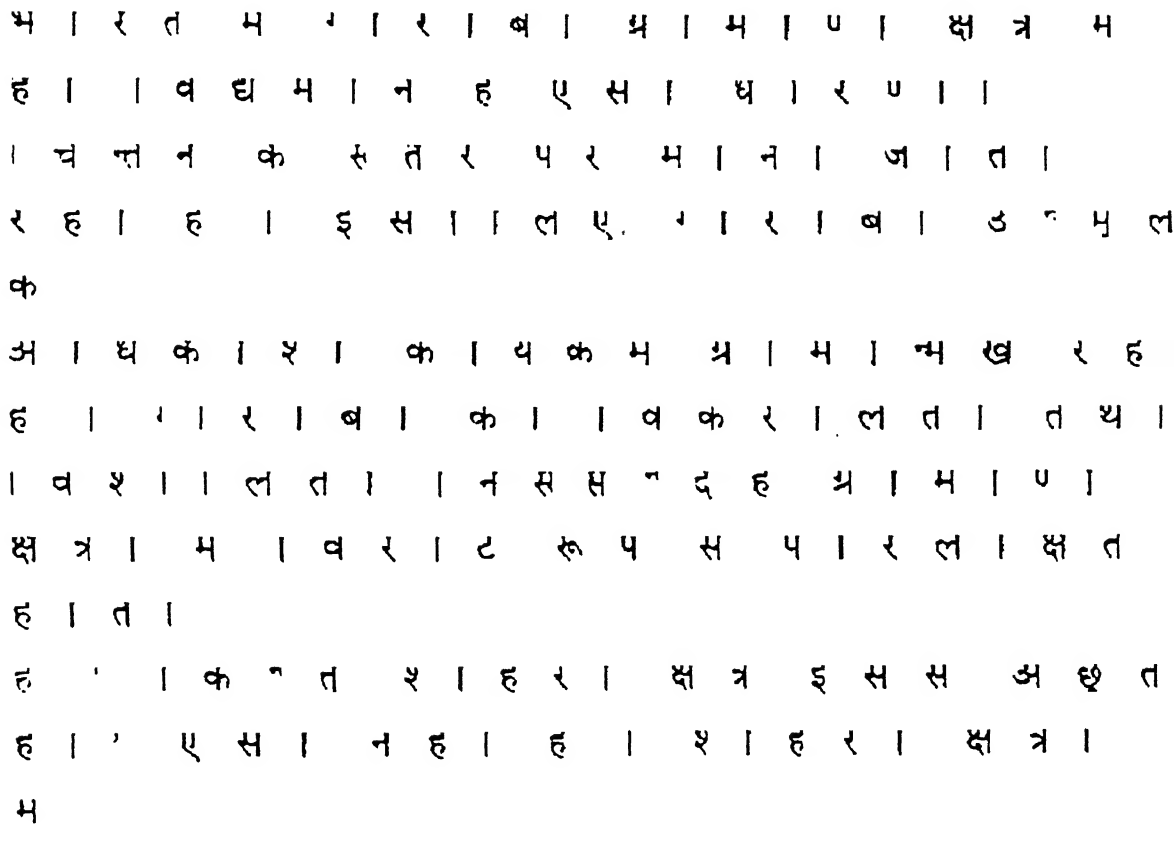


Figure 67: The image after conjunct segmentation (lines 1-5)

Revised Output of IOCR

भारत में गरीबी यामीण क्षेत्र मे ही वियमान है ऐसी धारणा
चिसन के स्तर पर मानी जाती रही है। इसीलिए गरीबी उव्यूलन के
अधिकांश कार्यकम ग्रामोव्मूख रहे है। गरीबी की विकरालता तथा
विशालता निस्प्र व्रेह सामीण क्षेवो मे विराट रुप से परिलक्षित होती
है, किचु शहरी क्षेब इससे अछूता हो, ऐसा नही है। शहरी क्षेवों में
गरीबी का रूप थिव है। शहरी क्षेव थी वनसंख्क्षा के अनुसार
अलग-अलग वेणी में वर्गीकृत हैं और इन क्षेवों में गरीबी का
बिस्तार अलग-अलग है। महानगरों सें जहां निम्चतम वर्ग के लोगों
की संख्सा अपेक्षाकृत यधिक होती है, वे तकनीकी रूप से गरीबों
की श्रेणी में रखे लाने के पाव नहीं होते क्पोंकि उनकी आय का स्तर
अ**कृत अभिक होगा। ऐसे महानगरों में मण्यम वर्ग का वतिशत
भी अपेक्षाकृत *ंचा है जिनकी आय *** रूपये प्रति माह से
**** रुपये प्रति माह होगी, लेकिन यह सतिशत छोटे नगरों या
कस्बों में भिव होगा। वहां आय का स्तर अपेक्षाकृत कम होगा और
वहां के निबाक्षियों में गरीबी रेखा के नीचे रहने वालों का प्रतिशत
ज्यादा होगा। लेकिन, यह वर्गीकरण गरीबी की थयाबहता को कम
नहीं कर पाता। वस्तुतः शहरी *त्रों में जो लोग विपवता की क्थिति
में जीवनयापन करते हैं, अनमें से अचिकांश गांबों से पलायन कर
आये हुए लोग हैं जो आते ही रोजगार की तलाश में हैं। दिनोंदिन
विस्तारित होती गंदी बस्तियों और फुटपाथों पर बसेरा करने वाले
जोग और अल्य दीवहीन बेरोजगार वर्ग के लोग शहरी गरीबी के
मठनक्ष रूप हैं।

Figure 68: The revised output of the classification process; only the top choice is shown here; * represents a rejected character.

Output afer post-processing the revised IOCR output

भारत में गरीबी ग्रामीण क्षेत्र मे ही विद्यमान है ऐसी धारणा
चिंतन के स्तर पर मानी जाती रही है। इसीलिए गरीबी उन्मूलन के
अधिकांश कार्यकम ग्रामोन्मुख रहे है। गरीबी की विकरालता तथा
विशालता निस्सन्देह ग्रामीण क्षेत्रों मे विराट रुप से परिलक्षित होती
है किन्तु शहरी क्षेत्र इससे अछूता हो ऐसा नही है। शहरी क्षेत्रों में
गरीबी का रूप { थिव } है। शहरी क्षेत्र भी जनसंख्या के अनुसार
अलग अलग श्रेणी में वर्गीकृत हैं और इन क्षेत्रों में गरीबी का
विस्तार अलग अलग है। महानगरों में जहां निम्नतम वर्ग के लोगों
की संख्या अपेक्षाकृत अधिक होती है वे तकनीकी रूप से गरीबों
की श्रेणी में रखे जाने के पात्र नहीं होते क्योंकि उनकी आय का स्तर
अपेक्षाकृत अधिक होगा। ऐसे महानगरों में मध्यम वर्ग का प्रतिशत
भी अपेक्षाकृत { कंचा } है जिनकी आय *** रूपये प्रति माह से
*** रुपये प्रति माह होगी लेकिन यह प्रतिशत छोटे नगरों या
कस्बों में { भिव } होगा। वहां आय का स्तर अपेक्षाकृत कम होगा और
वहां के निवासियों में गरीबी रेखा के नीचे रहने वालों का प्रतिशत
ज्यादा होगा। लेकिन यह वर्गीकरण गरीबी की भयावहता को कम
नहीं कर पाता। वस्तुतः शहरी क्षेत्रों में जो लोग { विपत्रता } की स्थिति
में जीवनयापन करते हैं उनमें से अधिकांश गांवों से पलायन कर
आये हुए लोग हैं जो आते ही रोजगार की तलाश में हैं। दिनोंदिन
विस्तारित होती गंदी बस्तियों और फुटपाथों पर बसेरा करने वाले
लोग और अन्य दीनहीन बेरोजगार वर्ग के लोग शहरी गरीबी के
प्रत्यक्ष रूप हैं।

Figure 69: The output of the classification process; the word has been underlined if the true word is the second or third choice; incorrect words are shown in { }; *** has been used for unr ecognized words; the numerals have not been incorporated in the system yet.

the other hand, Devanagari script being a two-dimensional composition of symbols, poses several problems for automating the training stage.

Although recognition performance of the system is dependent upon the discrimination capability of the features that constitute the prototype and the decision logic employed during recognition stage, the inadequate and/or inaccurate training contribute significantly to the reduction of its performance.

Some of the desirable features of an automated trainer are as under:

- It must establish correct correspondence of the unknown symbol with the true symbol. All spurious unknown symbols be ignored.
- It should present the unknown symbols in the same form as obtained during the process of recognition rather than the idealized symbols. This is a very important aspect as in practice, OCR segmentation and decomposition algorithms are rarely perfect and do introduce aberrations of its own.
- In case of touching symbols or fragmented symbols, the unknown symbol boxes do not match with the count of true boxes. An easy way out is to ignore such words altogether. However this leads to inadequate training besides loosing valuable information about how they physically appear in real-life. The trainer must not ignore such words and attempt to do the best possible.
- The trainer must prompt about inadequacies of the training, so that the designer may devise at the recognition stage necessary steps to cope up with this. For example, if the training set has very few or no sample for a symbol, any distance criterion based on say Mahalnobis distance will be erroneous. It will be more appropriate to use structural properties in such a case. The trainer must generate sufficient information to guide the decision making process.

Let us examine the aspects that need to be emphasized for designing an automated trainer for construction of prototypes for Devanagari script text recognition.

In real-life texts, characters may be fused, be under shadow of another character or fragmented. In addition there may be natural breaks. As explained earlier chapters, it is not possible to treat each conjunct or composite character as a separate atomic symbol as such combinations are very large in number.

The segmentation process outlined in 4 is used by the trainer also for obtaining symbols and characters. The training process uses same algorithms for obtaining the symbols/characters. The composite characters are identified after preliminary segmentation. In the recognition phase, The difference is in the way symbols/characters are accepted for building prototypes. A word in Devanagari script needs to be first segmented into what may be called as composite characters and then each composite character needs to be decomposed into a set of symbols. These symbols may correspond to a Devanagari character, a modifier symbol or may not have any meaning when viewed in isolation (this normally happens when the characters get fragmented or a modifier symbol extends from upper to core-strip region of the word). It is these decomposed symbols for which the trainer is required to construct the prototypes for the recognition stage.

In order to design an automated trainer, we need to construct a file of "true" data file which correspond to the sample text. While for Roman, it is simply an ASCII file which represent the individual characters of Roman script, in Devanagari this file in the form of ISCII (Indian Script Code for Information Interchange) codes. However, the ISCII codes do not correspond to the symbols as is obtained through the process of segmentation and decomposition as outlined earlier. On the other hand, if one wants to train the system for constructing prototypes of these symbols interactively, the corresponding symbols must be entered in some coded form after linearization and visual recognition. This is a highly strenuous job which is slow and prone to errors. Moreover, a user at the site is not expected to know about how to train the system on a new font etc. Thus we see that there is a need to design an automated trainer for Devanagari script which would take input from an ISCII file and automatically establish correspondences of the constituent symbols with those of physical symbol boxes obtained by optical process.

7.3 The Training Process and its Automation

The Training Process

(a) Input text words:

सुनिश्चित निर्णय

(b) Sample true-file symbols as represented by ISCII code (Phonetic order):

स ु न ि श ् च ि त न ि र ् ण य

(c) Physical symbol boxes obtained through optical processing of words of (a) assuming no fusion or fragmentation:

Upper region:
Core-strip: स ा न ा श च त ा न ण ा य
Lower region: ु

(d) Rearrangement of symbols of (b) into visual-order after applying composition rules:

(Composition rules used in the example: श ् = श्र, र ् ण = र्ण)

सुनिश्चित निर्णय

(e) Delineation of different regions of (d) above:

Upper region:
Core-strip: स ा न ा श च त ा न ण ा य
Lower region: ु

Figure 70: An Example illustrating the training process: (a) Input text words; (b) Sample true-file symbols as represented by ISCII code (Phonetic order); (c) Physical symbol boxes obtained through optical processing of input words; (d) and (e) show the stages through which the true file needs to be transformed to provide one-to-one correspondence of symbols for training. The example is under the assumption that there are no fusions or fragmentations.

Figure 70 presents an illustration of the process of training for construction of

prototypes for Devanagari. The trainer has two inputs: the scanned input data and the true-file text internally represented in ISCII code which is obtained after linearization of characters and symbols in the phonetic order [70]. These correspond to as given in figure 70(a) and 70(b) respectively. As outlined earlier, the input scanned data is segmented into lines and lines into words. Thereafter each word is attempted to be segmented and decomposed into individual symbols. This process is hindered by character shadowing, character fusion and character fragmentation. Hence the number of physical symbol boxes may not correspond to the expected number of boxes. However, if there are no fusions or segmentations, then the symbol boxes obtained for the example are as shown in figure 70(c). As shown the symbol boxes belong to three strips called upper modifier region, core-strip region and lower modifier region. The arrangement of the symbol boxes is that of visual order. The task of the trainer now is to establish correspondence between these physical symbol boxes (figure 70(c)) with those which correspond to true file (figure 70(b)). Firstly, the true file which is in phonetic order of symbols for internal representation, is rearranged to visual order using a set of rules for transformation [71] and script composition rules [71]. This is shown in figure 70(d). Thereafter, this is further delineated into the upper modifier, core-strip and lower modifier regions as shown in figure 70(e). Now we observe that there is one to one correspondence with the OCR physical symbol boxes as shown in figure 70(c) and those of figure 70(e) obtained from the true file. The process of construction of prototypes can begin now.

However, the trainer has to do more work in case of character fusion, shadowing or fragmentation. The number of physical symbol boxes gets reduced in case of fusion and shadowing, while it gets increased under fragmentation in comparison with the expected number of physical boxes as per the true text file. It is likely that the word under consideration has both the fusion and fragmentation such that there is a perfect match between the number of boxes. This happens rarely and we do not consider this case. In cases of mismatch, we do not know which boxes have actually fused or fragmented within the word. Segmentation is attempted only for those boxes which are suspected to be fused or shadowed. Box width is one of the major criterion used here. However, the character width in case of

Devanagari varies a lot and has to be estimated from the sample data. As the boxes are susceptible to inaccurate segmentation, further confirmation is required before establishing correspondences between the physical boxes and the true symbol boxes to avoid incorrect correspondence. These are achieved using a two-pass training schema as outlined in the following section.

Two-Pass Training Schema

For automating the training process for construction of prototypes for Devanagari, we employ a two-pass schema. In the first pass, the trainer as usual extracts the lines from the text and finds the word boundaries. If the number of words in the input text does not match with the expected number of words as per the true file, the whole line is ignored for training during this pass. For each word, an elementary segmentation process (which does not expect character fusion or fragmentation) is applied. The words are further divided into upper, lower and core-strip region as outlined earlier. This yields us a number of physical symbol boxes corresponding to each word. At this stage if there is no mismatch in the number of symbol boxes as obtained physically and the expected number derived from the true file, information about width and height of the characters/symbols are collected and analyzed statistically for use in the second pass. In addition, the prototypes are constructed for these symbols.

In the second pass, we concentrate on those words which have earlier shown a mismatch on the number of symbol boxes. A number of heuristics are used to ascertain the plausible fused or shadowed boxes. Similarly, after the segmentation also, a number of constraints are applied before accepting the box correspondences for the training purpose.

Firstly, the box must be relatively wide to qualify for the segmentation. The width threshold is obtained through the statistical analysis of Pass-I data. Further, we use vertical bar and box position properties. Both of these features are independent of font or size of the character and are reliable. These can be visually extracted for

the character set.

The box suspected to be fused must exhibit a combination of these two properties obtained through fusion of constituent characters as per the true text file. Thus a box is hypothesized to be fused if the above mentioned constraints are met. After the segmentation, once again these two properties are sought to be matched between the physical segmented box and the symbols as per the true file. If it fails, then the next wide box within the word is explored in a similar fashion for possible fusion. At the end of the process, all the physical symbol boxes must satisfy the above mentioned two properties and the width constraints as expected as per the true text file of the sample text. Otherwise the word under consideration is ignored for the purpose of training.

The case of character fragmentation is dealt with in a similar fashion. The hypothesized fragmented boxes are simply put together and all the above mentioned constraints are applied.

In our implementation and testing, we have found that the two pass schema as outlined above, works very well and establishes correct correspondences in all the cases for which the word is accepted for training. We have trained our system on two different fonts using a training set of 2000 characters for each font.

7.4 Experimentation

The system for reading Devanagari script document printed in Hindi language has been implemented on a DEC-2000 system. The code consists of more than 10,000 lines of C code which has been broken into 50 files in the form of functions. A serious effort has been made to develop a modular system. Addition of a new features requires almost no change anywhere. Interface is clean because all interactions take place through the solution blackboard.

Our system uses only font specific features when the font is known and the corresponding prototype library is available. Otherwise, the font-independent features are used which include vertical bar property, modified horizontal crossings vector and structural representation. We present the results in the next section for both the experimentations.

7.4.1 System Performance

We tested our Devanagari document reading system on various printed documents and gathered various results. In the first run, the moments feature, which is a font specific feature, is used. The performance at character level for font-I is shown in table 21 before the post-processing. The word level performance after post-processing is shown in table 22. The threshold distance for terminating the search is 3.0 in this experiment. The threshold is changed to 1.0 and the experiment is repeated. The results which show an improvement in the word level recognition are shown in table 23. The performance at character level before and after post-processing phase is compared in table 24 for 1.0 thresholds. The performance at word level before and after post-processing phase is compared in tables 25 and 26 for 3.0 and 1.0 thresholds. The performance for the conjuncts before and after post-processing phase is compared in tables 27 and 28 for 3.0 and 1.0 thresholds. The results for font II are presented only for threshold distance 1.0 for terminating the search. The results are presented in tables 29, 30, 31, 32 and 33.

	number of chars.	RECOGNITION				ERROR	
		top choice	in next two choices	in subsequent two choices	in remaining choices	substitutions	rejections
Doc I	812	735	30	4	6	34	3
Doc II	626	456	73	25	13	58	1
Doc III	527	386	41	13	14	72	1
Doc IV	465	387	29	7	8	33	1
Doc V	674	523	59	6	14	69	3
Doc VI	839	665	77	21	16	51	9
Doc VII	762	685	42	7	4	23	1
Doc VIII	898	721	93	18	15	46	5
Doc IX	756	664	49	3	7	32	1
Doc X	664	501	81	20	28	34	0
Overall	7023	5723	574	124	125	452	25
		81.48%	8.17%	1.76%	1.77%	6.43%	.35%

Table 21: Performance of the system at character level before the post-processing phase Font I.

Doc. No.	total words	Dictionary Words						Non-Dictionary Words			
		dic words	correct words	Error				non-dic words	correct words	incorrect words	forced to map to a a dic. word
				total words	2 char. words	3 char. words	≥ 4 char. words				
I	247	246	220	26	13	5	8	1	1	0	1
II	216	204	154	50	31	16	3	12	2	10	0
III	158	131	100	31	20	7	4	27	11	16	16
IV	133	126	106	20	12	8	0	7	4	3	4
V	212	190	154	36	15	16	5	22	7	15	7
VI	247	222	198	24	13	8	3	25	10	15	8
VII	247	227	208	19	10	5	4	20	18	2	16
VIII	292	255	212	43	21	13	9	37	25	12	25
IX	218	207	191	16	13	0	3	11	8	3	8
X	220	206	151	55	37	15	3	14	10	4	10
Overall	2190	2014	1694	320	185	93	42	176	96	80	95
			84.11	15.88%	9.18%	4.61%	2.08%		54.54%	45.45%	53.97%

Table 23: Performance of the system at word level using word level knowledge at threshold distance 1.0(Font I).

	number of chars.	correct recognition before using word dictionary	correct recognition after using dictionary	improvement
Doc I	812	735	748	13
Doc II	626	456	503	47
Doc III	527	386	407	21
Doc IV	465	387	409	22
Doc V	674	523	552	29
Doc VI	839	665	753	88
Doc VII	762	685	704	19
Doc VIII	898	721	752	31
Doc IX	756	664	705	41
Doc X	664	501	535	34
Overall	7023	5723	6068	345
		81.48%	86.40%	4.91%

Table 24: Comparison of the performance of the system at character level before and after using word level knowledge (Font I) at 1.0 distance threshold for terminating the search

	number of words	top word before consulting the dictionary	top word after consulting the dictionary	improvement
Doc I	244	179	210	31
Doc II	202	88	142	54
Doc III	151	79	106	27
Doc IV	128	71	100	29
Doc V	203	108	154	46
Doc VI	211	105	146	41
Doc VII	246	182	222	40
Doc VIII	288	165	220	55
Doc IX	216	149	186	37
Doc X	208	107	133	26
Overall	2097	1233	1619	386
		58.79%	77.20%	18.40%

Table 25: Comparison of the performance of the system at word level, before and after looking up the dictionary (Font I) when distance threshold is 3.0 for stopping the search.

	number of words	top word before consulting the dictionary	top word after consulting the dictionary	improvement
Doc I	247	192	221	29
Doc II	216	92	162	70
Doc III	158	85	116	31
Doc IV	133	75	108	33
Doc V	212	113	167	54
Doc VI	247	122	202	80
Doc VII	247	187	215	28
Doc VIII	292	172	224	52
Doc IX	218	152	199	47
Doc X	220	112	151	39
Overall	2190	1302	1765	463
		59.45	80.59%	21.14%

Table 26: Comparison of the performance of the system at word level, before and after looking up the dictionary (Font I) when distance threshold is 1.0 for stopping the search.

Doc. no.	total chars.	touching chars.	Before post processing: top choice	Before post processing: subsequent 3 choices	Before post processing: substitution	Before post processing: rejection	After post processing: top choice	After post processing: subsequent 3 choices	After post processing: substitution	After post processing: rejection
I	803	38	22	3	13	0	30	3	5	0
II	604	40	16	2	22	0	24	6	10	0
III	507	28	9	1	16	2	14	6	6	2
IV	451	20	13	2	5	0	16	3	1	0
V	649	48	26	5	17	0	31	7	10	0
VII	764	48	32	6	9	1	44	0	4	0
VIII	887	48	20	7	20	1	30	6	11	1
IX	751	42	26	8	8	0	31	1	10	0
X	634	22	17	0	1	4	16	2	4	0
Overall	6919	334	181	35	111	8	236	34	61	3
		4.82%	54.19%	10.47%	33.23%	2.39%	70.65%	10.17%	18.26%	.89%

Table 27: Performance of the system for conjuncts, before and after post-processing (Font I) when distance threshold is 3.0 for stopping the search.

Doc. no.	total chars.	touching chars.	Before post processing				After post processing			
			top choice	subsequent 3 choices	substitution	rejection	top choice	subsequent 3 choices	substitution	rejection
I	812	38	29	2	2	5	33	1	4	0
II	626	40	28	1	1	10	34	1	3	2
III	527	28	10	3	2	13	14	0	6	8
IV	465	26	19	3	0	4	23	2	1	0
V	674	50	38	1	0	11	37	3	6	4
VI	839	39	22	9	7	1	30	0	9	0
VII	762	44	37	4	0	3	40	0	4	0
VIII	898	46	28	4	4	10	35	0	4	7
IX	756	42	26	8	8	0	39	1	2	0
X	664	30	19	2	9	0	20	2	8	0
Overall	7023	383 5.45%	256 66.84%	36 9.39	33 8.61%	58 15.14%	305 79.63	10 2.61	47 12.27%	21 5.48%

Table 28: Performance of the system for conjuncts, before and after post-processing (Font I) when distance threshold is 1.0 for stopping the search.

	number of chars.	RECOGNITION				ERROR	
		top choice	in next two choices	in subsequent two choices	in remaining choices	substitutions	rejections
Doc I	1193	954	79	29	37	84	10
Doc II	1305	1044	78	24	18	108	33
Doc III	1129	896	82	17	28	91	15
Doc IV	1368	1135	98	25	16	82	12
Doc V	1069	874	60	9	31	88	7
Doc VI	605	487	40	15	34	25	4
Doc VII	900	714	70	30	48	31	7
Doc VIII	723	540	72	36	28	41	6
Doc IX	441	366	26	15	11	21	2
Doc X	637	535	31	14	11	41	5
Overall	9370	7545	636	214	262	612	94
		80.52%	6.78%	2.28%	2.79%	6.53%	1.00%

Table 29: Performance of the system at character level before the post-processing phase Font II.

Doc. No.	total words	correct words	Error			
			total words	2 char. words	3 char. words	≥ 4 char. words
Doc I	382	320	62	39	16	7
Doc II	426	362	64	41	17	6
Doc III	383	324	59	42	12	5
Doc IV	464	392	72	38	21	13
Doc V	381	319	62	45	13	4
Doc VI	191	160	31	20	7	4
Doc VII	400	344	56	42	8	6
Doc VIII	221	183	38	25	8	5
Doc IX	85	71	14	8	4	2
Doc X	210	178	32	17	8	7
Overall	3143	2653	490	317	114	59
		84.40%	15.59%	10.08%	3.62%	1.87%

Table 30: Performance of the system at word level using word level knowledge at threshold distance 1.0(Font II).

	number of chars.	correct recognition before using word dictionary	correct recognition after using dictionary	improvement
Doc I	1193	954	1061	107
Doc II	1305	1044	1122	78
Doc III	1129	896	965	69
Doc IV	1368	1135	1176	41
Doc V	1069	874	879	5
Doc VI	605	487	526	39
Doc VII	900	714	774	60
Doc VIII	723	540	621	81
Doc IX	441	366	374	8
Doc X	637	535	546	12
Overall	9370	7545	8045	500
		80.72%	85.85%	5.33%

Table 31: Comparison of the performance of the system at character level before and after using word level knowledge (Font II) at 1.0 distance threshold for terminating the search.

	number of words	top word before consulting the dictionary	top word after consulting the dictionary	improvement
Doc I	382	240	320	80
Doc II	426	266	362	96
Doc III	383	271	324	73
Doc IV	464	222	392	70
Doc V	381	236	319	83
Doc VI	191	103	160	57
Doc VII	400	243	344	101
Doc VIII	221	135	183	48
Doc IX	85	49	71	22
Doc X	210	136	178	42
Overall	3143	1748	2653	463
		55.61%	84.40%	28.79%

Table 32: Comparison of the performance of the system at word level, before and after looking up the dictionary (Font II) when distance threshold is 1.0 for stopping the search.

7.4.2 Pitfalls and Limitations

While compiling the recognition results, we learnt several reasons for recognition errors. The fading near header line where the top modifiers are joined with the header line, results in wrong top boundary for the word causing the top modifiers to be left out. In the same manner, the lower modifiers are left out. In both the cases, the word level recognition suffers. Sometimes, a character is divided into two character boxes due to the fading.

Sometimes, the segmentation algorithm fails to segment a conjunct. The conjunct segmentation algorithm relies on the structural properties of the characters which get violated due to fading. Cases have also been observed when a conjunct is classified to a wrong character class with high enough confidence resulting in substitution error.

For *short* words, many competing words are found in the dictionary. The selection based on the distance is not always correct. A *short* word from which top/lower modifiers have been left out due to wrong word boundary, is searched in the wrong partition. For other words, we assume that at least one alias will contain two true characters. We observed certain cases when this condition was violated and the word was searched in the wrong partition. Sometimes, this condition was met, but an alias searched earlier caused the search to terminate. In both the cases, true word was lost.

Doc. no.	total chars.	touching chars.	Before post processing				After post processing			
			top choice	subsequent 3 choices	substitution	rejection	top choice	subsequent 3 choices	substitution	rejection
Doc I	1193	49	33	5	11	0	36	3	10	0
Doc II	1305	72	40	12	20	0	56	8	8	0
Doc III	1129	64	40	9	14	1	53	3	8	0
Doc IV	1368	60	28	4	28	0	44	8	8	0
Doc V	1069	60	44	2	14	0	50	0	10	0
Doc VI	605	32	20	5	6	1	26	1	4	1
Doc VII	900	46	31	4	7	4	37	3	4	2
Doc VIII	723	36	22	8	4	2	26	3	6	1
Doc VIII	441	18	10	2	4	2	12	2	4	0
Doc VIII	637	32	20	4	6	2	24	3	5	0
Overall	9370	469 5.00%	288 61.40%	55 11.72%	114 24.30%	12 2.55%	364 77.61%	34 7.24%	67 14.28%	4 0.85%

Table 33: Performance of the system for conjuncts, before and after post-processing (Font II) when distance threshold is 1.0 for stopping the search.

Chapter 8

Conclusions

We have designed and implemented an integrated system for Devanagari text recognition. We have integrated relevant knowledge kources with various modules of the system. Some of the knowledge kources are generated from the document during its processing. Such knowledge kources are transient in nature and have relevance for the processing of the current document session. For instance, the text height, the character width and height information constitute transient Knowledge Sources which guide the text line and character segmentation respectively. Some knowledge kources are specific to a font and become ineffective in the context of a different font. The moment vector feature and aspect ratio are two of the character classifying features which are specific to a font. We use these knowledge kources only when the font is known. There are many knowledge kources which are robust and remain effective for a large number of fonts. The classification of characters in three classes is based on the coverage of the core strip is robust. Further classification of these classes based on presence and position of the vertical bar is also robust. The modified horizontal zero crossings vector is also a robust classifying feature. The envelop information is used for generating character hypotheses from the word dictionary. The structural properties and descriptions are also robust for the reduced set obtained through a filtering process. The character descriptions are altered due

to the presence of noise due to ink spread or aberrations due to segmentation. We handle the variations by extensive training which exposes the system to natural variants of characters. The vertical bar property is used as a reference in structural descriptions which keeps the number of distinct prototypes for a character under check. Also, a stroke whose length is less than 1/3rd of minimum of height and width of the character is ignored as noise.

During the training phase, we collect the character classes which classification process tends to confuse. The character classification matrix is used by the post-processing phase for giving preference to a known confusion while substituting a character by another one for mapping a word to a dictionary word. The dictionary words have been hierarchically divided using word envelop information consisting of number of the *upper, lower* and *core* modifiers along with number of core characters. and *tags*. A *tag* is a string of fixed length associated with each partition. During the matching process, a mapping for an unknown character is allowed only if it has been classified with low confidence.

A confidence figure is associated with every character by the classification process. The confidence figure is inversely proportional to the *distance* between the stored prototype of the known character and the one obtained from the unknown character. The candidate characters are ranked based on the cumulative *distance* obtained after applying all the features. If the top ranked character is a low confidence character, the character box becomes a candidate for further segmentation.

The knowledge kources are integrated together in an architecture based on the blackboard model. The system consists of a solution blackboard and various knowledge sources. There is a set of tools that is used for applying the knowledge sources. The initial control program invokes modules to segment each uniform text zone into text lines and words. The image of the word is further segmented into easily separable units and are posted on the blackboard. The middle level control program is then activated by the initial control program after symbol image boxes are posted on the blackboard. It returns control to the initial control program after processing all

the character images posted on the blackboard. Here, the height and width of the character boxes of a text line are analyzed and the threshold height and width are set. The character boxes which are higher than threshold height are conjectured to contain a lower modifiers and which are wider than threshold width are conjectured to contain a conjunct. The middle level control program invokes an appropriate control process based on the type of the image box (core character, upper modifier, lower modifier). The confidence figure associated with each hypothesis is checked. If none of the candidate characters has acceptable confidence figure and the image is conjectured to be a composite character image, further segmentation is done. The segmented image is appended on the blackboard. After processing every image constituting a word, the output is composed back into the word(s). Words are then checked in the language dictionary and corrected if necessary. We have also used a character pair expert to disambiguate character pairs such as म स, प य, ज न etc. The character pair expert looks at a predetermined region of the image based on the pair under consideration and looks for specific properties. We have found it the character pair expert to be quite effective in case of closely resembling characters.

We have tested our system on two different fonts of printed text and achieved a promising performance of 81% at character level and 60% at word level prior to the post-processing phase. This performance is further enhanced by using word dictionary to 85% at character level and 80% at word level. Out of all conjuncts, 82% conjuncts have been segmented and 12% remains unsegmented. The performance dropped to 70% at the character level when only the font independent classifying features were used.

Bangla OCR

An OCR for Bangla (second most popular script and language in the Indian subcontinent) printed script has been described by Chaudhuri and Pal [100, 101, 102]. The emphasis is on recognizing the vowels and consonants (basic characters) which constitute 94-96% of the text. The remaining are compound characters which result from compounding of two (sometimes, even three) characters such that the resultant

shape is different from the shapes of constituent characters. The basic characters are described in terms of nine types of strokes. This approach has been tried for ideographic languages. A tree classifier is used to check the presence of each of these strokes and narrow down the set of candidate characters at each node. In addition, some other simple features, specific to the character or character pair are used to make a decision. The strokes are described in terms of their lengths with respect to the width and/or height of the minimum upright bounding box of the character image under consideration. The expected angle of each stroke is also defined (45 deg, 315 deg etc.).

These angles are very font specific and will vary from one font to another. Moreover, the fading of some black pixels may mislead the stroke extraction process. Even the hand crafted character specific features may not sustain their utility across various fonts. Even if a single document is assumed to have only one font, the user should be able to train the OCR on another font and use it. This Bangla OCR uses very specific font information for extracting the features and for disambiguating between character pairs. For compound characters, template matching is being used. The correlation measures given by Jaccard and Needham is employed [103].

They have not presented any sample document and corresponding output for various stages. The quality of their test documents remain hidden. Results are summarized in one statement which do not tell anything about the results at various stages and the contribution of each stage in a comprehensive manner.

The pre-processing stage includes noise cleaning which is histogram-based thresholding. The skew detection and correction are based on an extension of earlier ideas on skew detection and correction where they use uppermost and lower most pixels of the connected components to estimate the skew angle. There is no new technique or approach described in this paper for character recognition.

References

[1] Franz L. Alt, *Digital Pattern Recognition By Moments*, in Optical Character Recognition, edts: G. L. Fischer et al, McGreger and Wreger, USA, 1962.

[2] H. Arakawa et al, *On-line recognition of handwritten characters-Alphanumerics, Hiragana, Katakana, Kanji*, Proceedings 4th International Joint Conference Pattern Recognition (IJCPR), pp 810-812, November 1978.

[3] V. Bansal and R. M. K. Sinha, *Designing a Front End OCR System for Indian Scripts for Machine Translation - A Case Study for Devanagari*, Symposium on Machine Aids for Translation and Communication (SMATAC-96), New Delhi, India, 1996.

[4] H. S. Baird, *Feature Identification for Hybrid Structural/Statistical Pattern Classification*, Computer Vision, Graphics and Image Processing, vol. 42, pp. 318-333, 1988.

[5] P. Bao-Chang, W. Si-Chang and Y. Guang-Yi, *A method of Recognizing Handprinted Characters*, Computer Recognition and Human Production of Handwriting, Eds. R. Plamondon, C. Y. Suen and M. L. Simner, World Scientific Publ. Co., pp. 37-60, 1989.

[6] T. Bayer et al, *Towards the Understanding of Printed Documents*, Structured Document Image Analysis, eds: H. S. Baird, H. Bunke, K. Yamamoto, Springer-Verlag, U.S.A., 1992.

[7] P. W. Becker and K. A. Neilsen, *Pattern Recognition using dynamic pictorial information*, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-2, pp. 434-437, 1972.

[8] R. M. Brown, *On-line recognition of hand-printed characters*, IEEE Transactions on Electron. Comput., vol. EC-13, pp. 750-752, Dec. 1964.

[9] D. J. Burr, *Elastic Matching of Line Drawings*, Proceedings International Conference on Pattern Recognition (ICPR), Florida, pp 223-228, 1980.

[10] R. G. Casey and D. R. Furgson, *Intelligent Forms Processing*, IBM System Journal, vol. 29, no. 3, pp. 435-450, 1990.

[11] G. C. Cash and M. Hatamian, *Optical character recognition by the method of moments*, Computer Vision, Graphics and Image Processing, vol. 39, pp. 291-310, 1987.

[12] M. Daneman, P. A. Carpenter, and M. A. Just, *Cognitive Processes and Skills*, Advances in Reading/Language Research, ed: B. A. Hutson, vol. 1, page 83-124, Jai Press, USA and England, 1980.

[13] Andreas Dengel, *ANASTASIL: A System for Low-Level and High-Level Geometric Analysis of Printed Documents*, Structured Document Image Analysis, eds: H. S. Baird, H. Bunke, K. Yamamoto, Springer-Verlag, U.S.A., 1992.

[14] R. Engelmore and T. Morgan, editors, *Blackboard Systems*, Addison-Wesley Publishing Company, Great Britain, 1988.

[15] A. S. Fox and C. C. Tappert, *On-line external word segmentation for handwriting recognition*, Proceedings International Symposium on Handwriting Computer Application, July 1987.

[16] Herbert Freeman, *Boundary Encoding and Processing*, Picture processing and psychopictories, Edited by B. S. Lipkin and Azriel Rosenfeld, Academic Press, New York and London, 1970.

[17] K. S. Fu, *Syntactic Methods in pattern recognition* Academic Press New York and London, 1974.

[18] G. Gaillat, *An on-line recognizer with learning capabilities*, Proceedings 2nd International Joint Conference Pattern Recognition (IJCPR), pp. 305-306, 1974.

[19] R. C. Gonzelez and P. Wintz, *Digital Image Processing*, 2nd ed. Reading, MA: Addison-Wesley, 1987.

[20] V. K. Govindan and A. P. Shivprasad, *Character Recognition - A Review*, Pattern Recognition, vol. 23, no. 7, pp 671-683, 1990.

[21] S. Hanaki et al, *On-line recognition of handprinted Kanji characters*, Pattern Recognition, vol. 12, pp 421-429, 1980.

[22] S. Hanaki et al, *An on-line character recognition aimed at a substitution for a billing machine keyboard*, Pattern Recognition, vol. 8, pp 63-71, 1976.

[23] W. J. Hannan, *RCA multifont reading machine*, Optical Character Recognition, eds: G. L. Ficher et al, McGregor and Wemer, pp. 3-14, 1962.

[24] S. Harmalkar and R. M. K. Sinha, *Integrating word level knowledge in text recognition*, 10th International Conference on Pattern Recognition, Atlantic city, New Jersy, 1990.

[25] W. Hattich, H. Tropf and G. Winkler, *Combination of statistical and syntactical pattern recognition-Applied to classification of unconstrained handwritten numbers*, Proceedings International Conference on Pattern Recognition (ICPR), Japan, pp 786-788, 1978.

[26] Barbara Hayes-Roth, *A Blackboard Architecture for Control*, Artificial Intelligence, vol. 26, pp 251-321, 1985.

[27] Barbara Hayes-Roth, *Intelligent Control*, Artificial Intelligence, vol. 59, pp 213-220, 1993.

[28] Barbara Hayes-Roth, *An Architecture for Adaptive Intelligent Systems*, Artificial Intelligence, vol. 72, pp 329-365, 1995.

[29] T. K. Ho, J. J. Hull and S. N. Srihari, *Decision combination in multiple classifier systems*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-16, no. 1, pp. 66-75, 1994.

[30] J. S. Huang and M. Lung, *Separating similar complex Chinese characters by Walsh transform*, Pattern Recognition, vol. 20, pp. 425-428, 1987.

[31] J. J. Hull and S. N. Srihari, *Experiment in text recognition with binary n-gram and viterbi algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-4, pp. 520-530, 1980.

[32] J. J. Hull, S. N. Srihari and R. Choudhari, *An integrated algorithm for text recognition: comparison with cascaded algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI- 5, 384-395, 1983.

[33] M. K. Hu, *Visual pattern recognition by moment invariants*, IRE Transactions on Information Theory, vol. IT-8, pp. 179-187, 1962.

[34] T. Ichikawa and J. Yoshida, *On-line recognition of handprinted characters with associative read-out of patterns in memory*, Proceedings 2nd International Joint Conference Pattern Recognition (IJCPR), pp. 206-207, Aug. 1974.

[35] K. Ikeda et al, *On-line recognition of hand-written characters utilizing positional and stroke vector sequences*, Proceedings 4th International Joint Conference Pattern Recognition (IJCPR). pp. 813-815, November 1978.

[36] S. Impedovo et al, *A Fourier descriptors set for recognizing nonstylized numerals*, IEEE Transactions on Systems, Man and Cybernetics. vol. SMC-8, pp. 640-645, Aug. 1978.

[37] S. Impedovo, *Plane curve classification through Fourier descriptors: An application to Arabic handwritten numeral recognition*, Proceedings 7th International Conference Pattern Recognition (ICPR), pp. 813-815, November 1978.

[38] M. A. Just and P. A. Carpenter, *A theory of reading: From eye fixation to comprehension*, Psychological Reviews, vol. 87, pp. 329-354, 1980.

[39] S. Kahan, T. Pavlidis and H. S. Baird, *On the recognition of printed characters of any font and size*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-9, no. 2, pp. 274-287, 1987.

[40] D. D. Kerrick and A. C. Bovik, *Microprocessor based recognition of handprinted characters from a tablet input*, Pattern Recognition, vol. 21, pp. 525-537, 1988.

[41] C. Kimpan et al, *Fine classification of printed Thai character recognition using the K-L expansion*, IEEE Proceedings vol. E-134, pp 257-264, 1987.

[42] D. E. Knuth, *The METAFONTBook*, AMS and Addison Wesley, USA, 1986.

[43] M. Kushnir et al, *An application of the Hough transform to the recognition of printed Hebrew characters*, Pattern Recognition, vol. 16, page 183-191, 1983.

[44] M. Kushnir and K. Matsumoto, *Recognition of hand-printed Hebrew characters using features selected in the Hough transform space*, Pattern Recognition, vol. 18, page 103-114, 1985.

[45] M. T. Y. Lai and C. Y. Suen, *Automatic recognition by Fourier descriptors and boundary encoding*, Pattern Recognition, vol. 14, pp. 383-393, 1981.

[46] L. Lam and C. Y. Suen, *Structural classification and relaxation matching of totally unconstrained hand-written zip code numbers*, Pattern Recognition, vol. 21, pp. 19-31, 1988.

[47] G. Lee, J. Lee and J. Yoo, *Multi-level post-processing for Korean character recognition using morphological analysis and linguistic evaluation*, Pattern Recognition, vol. 30, pp. 1347-1360, 1997.

[48] Su Liang, M. Shridhar and M. Ahmadi, *Segmentation of Touching Characters in Printed Document Recognition*, Pattern Recognition, vol. 27, no. 6, pp. 825-840, 1994.

[49] E. Mandler et al, *Experiments in on-line script recognition*, Proceedings 4th Scandinavian Conference on Image Analysis, pp. 75-86, June 1985.

[50] S. S. Marwah, S. K. Mullick and R. M. K. Sinha, *Recognition of Devanagari characters using a hierarchical binary decision tree classifier*, IEEE International Conference on Systems, Man and Cybernetics, October 1994.

[51] S. Mori et al, *Historical Review of OCR Research and Development*, Proceedings IEEE , vol. 80, no. 7, pp. 1029-1058, July 1992.

[52] G. Nagy and S. Seth, *Hierarchical Representation of Optically Scanned Documents*, Proceedings 7th International Conference on Pattern Recognition (ICPR), Montreal, 1984.

[53] R. Narsimhan, *Labelling schemata and syntactic descriptions of pictures*, Information and Control, vol. 7, no. 2, pp. 151 - 179, June 1964.

[54] R. Narsimhan, *On the description, generation and recognition of classes of pictures* in Automatic interpretation and classification of images, ed. A. Grasseli, Academic press, chapter 1, New York, 1969.

[55] D. L. Neuhoff, *The Viterbi Algorithm as an aid in text Recognition*, IEEE Transactions on Information Theory, vol. IT-21, pp. 222 - 226, 1975.

[56] K. Odaka et al, *On line recognition of hand written characters by approximating each stroke with several points*, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-12, pp. 898-903, 1982.

[57] R. Ott, *On feature selection by means of principal axis transform and nonlinear classification*, Proceedings International Joint Conference on Pattern Recognition (IJCPR), pp. 220-222, 1974.

[58] E. Persoon and K. S. Fu, *Shape determination using Fourier descriptors*, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-7, 170-179, 1977.

[59] J. L. Peterson, *Computer program for detecting and correcting spelling errors*, Communications ACM, vol. 23, pp. 667-687, 1980.

[60] P. J. Pobgee et al, Application of a low cost graphical input tablet, Information Processing, vol. 71, pp. 738-741, 1972.

[61] R. S. Prawat, *Relations between Semantic Memory and Reading*, Advances in Reading/Language Research, ed: B. A. Hutson, vol. 1, page 51-81, Jai Press, USA and England, 1980.

[62] E. M. Riseman and R. W. Ehrich, *Contextual Word Recognition using binary diagrams*, IEEE Transactions on Computer, vol. C-20, pp. 397-403, 1971.

[63] E. M. Riseman and A. R. Hanson, *A Contextual Post Processing System for Error Correction using binary n-grams*, IEEE Transactions on Computer, vol. C-23, pp 480-493, 1974.

[64] Jairo Rocha and Theo Pavlidis, *A Shape Analysis Model with Applications to a Character Recognition System*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-16, no. 4, April 1994.

[65] J. C. Sant and S. K. Mullick, *Hnadwriten Devanagari script recognition using CTNNSE algorithm*, International Conference on Application of Information Technology in South Asian Language, February 1994.

[66] K. M. Sayre, *Machine Recognition of Handwritten Words- A Project Report*, Pattern Recognition, vol. 5, pp 213-228, 1973.

[67] A. C. Shaw, *Parsing of graph-representative pictures*, J. Ass. comput. Mach., vol. 17, pp. 345 - 362, 1969.

[68] R. M. K. Sinha and H. N. Mahabala, *Machine recognition of Devanagari script*, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-9, 435-441, 1979.

[69] R. M. K. Sinha, *PLANG-A picture language schema for a class of pictures*, Pattern Recognition, vol. 16, no. 4, pp. 373-383, 1983.

[70] R. M. K. Sinha, *Computer processing of Indian languages and scripts - potentialities and problems*, Journal of Institution of Electronics & Telecommunication Engineers, vol. 30, 133-49, 1984.

[71] R. M. K. Sinha, *Rule based contextual post-processing for Devanagari text recognition*, Pattern Recognition, vol. 20, no. 5, pp. 475-485, 1987.

[72] R. M. K. Sinha, *Visual text recognition through contextual processing*, Pattern Recognition, vol. 21, no. 5, pp. 463-479, 1988.

[73] R. M. K. Sinha, *On partitioning a dictionary for visual text recognition*, Pattern Recognition, vol. 23, no. 5, pp. 497-500, 1989.

[74] R. M. K. Sinha, *Standardizing Linguistic Information - An Overview*, Proceedings of Second Regional Workshop on Computer Processing of Asian Languages, Tata McGraw-Hill, New Delhi, pp 272-290, 1992.

[75] R. M. K. Sinha, B. Prasada, G. Houle and M. Sabourin, *Hybrid Contextual Text Recognition with String Matching*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-15, pp. 915-925, 1993.

[76] T. Sakai, *Several approaches to development of on-line handwritten character input equipment*, Proceedings 7th International Conference on Pattern Recognition (ICPR), pp. 1052-1054, 1984.

[77] A. Sanfeliu and K. Fu, *A Distance measure between attributed relational graphs for pattern recognition*, IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-13, no. 3, pp. 353-362, 1981.

[78] G. P. R. Sarvarayudu and I. K. Sethi, *Walsh descriptors for polygonal curves*, Pattern Recognition, vol. 16, pp. 327-336, 1983.

[79] I. K. Sethi, *Machine recognition of constrained handprinted Devanagari*, Pattern Recognition, vol. 9, pp. 69-75, 1977.

[80] I. K. Sethi and B. Chatterjee, *Machine recognition of handprinted Devanagari Numerals*, Journal of Institution of Electronics and Telecommunication Engineers, India, vol. 22, pp 532-535, 1976.

[81] S. C. Shapiro et al., Eds, *Encyclopedia of Artificial Intelligence*, vol. 1. Wiley Interscience, New York, 1987.

[82] L. Shapiro and R. Haralick, *Structural Description and Inexact Matching*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-3, pp. 504-519, September 1981.

[83] R. Shinghal and G. T. Toussaint, *Experiments in text recognition with modified Viterbi algorithm,* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, pp. 184-192, 1979.

[84] R. Shinghal, *A hybrid algorithm for contextual text recognition*, Pattern Recognition, vol. 16, pp. 261-267, 1983.

[85] M. Shridhar and A. Badreldin, *High accuracy character recognition algorithm using Fourier and topological descriptors*, Pattern Recognition, vol. 17, pp 515-523, 1984.

[86] B. Simard, B. Prasad and R. M. K. Sinha, *On-line character recognition using handwriting modeling*, Pattern Recognition, vol. 26, no. 7, pp. 993 - 1007, 1993.

[87] S. N. Srihari, *Integrating diverse knowledge sources in text recognition*, ACM Transactions on Office Information System, vol. 1, pp. 68-87, 1983.

[88] C. Y. Suen, *n-gram statistics for natural language understanding and text processing*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol PAMI-1, pp. 164-172, 1979.

[89] H. Takahashi et al, *A Spelling Correction Method and its Application to an OCR system*, Pattern Recognition, vol. 23, pp 363-377, 1990.

[90] C. C. Tappert, *Adaptive online handwriting recognition*, Proceedings 7th International Conference on Pattern Recognition (ICPR), pp. 1004-1007, 1984.

[91] C. C. Tappert, C. Y. Suen and T. Wakhara, *The state of the art in online handwritten character recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-12, no. 8, pp. 787-808, 1990.

[92] T. Taxt, P. J. Flynn and A. K. Jain, *Segmentation of document image*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-11, pp. 1322-1329, 1989.

[93] G. T. Toussaint, *The Relative Neighbourhood Graph of a finite planar set*, Pattern Recognition, vol. 12, pp. 261-268, 1980.

[94] S. Tsujimoto and H. Asada, *Resolving Ambiguities in Segmenting Touching Characters*, Structured Document Image Analysis, eds: H. S. Baird et al, Springer-Verlag, U.S.A., 1992.

[95] S. Wendling et al, *Use of Harr transform and some of its properties in character recognition*, Proceedings International Joint Conference on Pattern Recognition (IJCPR), pp. 844-848, 1976.

[96] K. Y. Wong, R. G. Casey and F. M. Wahl, *Document Analysis System*, IBM Journal of Research and Development, vol. 26, no. 6, pp. 647-656, 1982.

[97] H. Yamada, *Contour DP matching method and its application to handprinted Chinese character recognition*, Proceedings International Conference on Pattern Recognition (ICPR), Canada, pp. 389 - 392, 1984.

[98] K. Yoshida and H. Sakoe, *Online handwritten character recognition for a personal computer system*, IEEE Transactions on Consumer Electronics, vol. CE-28, pp. 202-209, August 1982.

[99] T. Y. Zhang and C. Y. Suen, *A fast parallel algorithm for thinning digital patterns*, Communications ACM, vol. 27, pp. 236-239, March 1984.

[100] B. B. Chaudhuri and U. Pal, *A Complete Printed Bangla OCR System*, Pattern Recognition, vol. 31, no. 5, pp. 531-549, 1997.

[101] B. B. Chaudhuri and U. Pal, *A Complete Printed Bangla OCR System*, Pattern Recognition, vol. 31, no. 5, pp. 531-549, 1997.

[102] U. Pal and B. B. Chaudhuri, *An improved Document Skew Angle Estimation Technique*, Pattern Recognition Letters, 17, pp. 899-904, 1996.

[103] J. D. Tubbs, *A Note on Binary Template Matching*, Pattern Recognition, vol. 11, pp. 207-222, 1986.

Papers written out of this work:

1. R.M.K.Sinha and V. Bansal , *On Devanagari Document Processing*, IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada 1995.

2. R.M.K. Sinha and Veena Bansal, *On automating trainer for construction of prototypes for Devanagari text recognition* Technical Report, TRCS-95-232, I.I.T. Kanpur, India, 1995.

3. Veena Bansal and R.M.K. Sinha, *On how to describe shapes of Devanagari characters and use them for recognition*, Technical Report, TRCS-96-241, I.I.T. Kanpur, India, 1996.

4. V. Bansal and R.M.K.Sinha,*On integrating diverse knowledge sources in optical reading of Devanagari script*, Proceedings of the international conference on Information Systems Analysis and Synthesis (ISAS-96), Orlando, USA, 1996.

5. Veena Bansal and R.M.K. Sinha, *Partitioning and Searching Dictionary for Correction of Optically-Read Devanagari Character Strings*, Technical Report, TRCS-97-246, I.I.T. Kanpur, India, 1997.

6. Veena Bansal and R.M.K. Sinha, *Segmentation of touching and fused Devanagari characters*, Technical Report, TRCS-97-247, I.I.T. Kanpur, India, 1997.

7. Veena Bansal and R.M.K. Sinha, *Integrating Knowledge Sources in Devanagari Text Recognition System*, Technical Report, TRCS-97-248, I.I.T. Kanpur, India, 1997.

Appendix A

Devanagari Script in Brief

Devanagari script is a logical composition of its constituent symbols in two dimensions. It is an alphabetic script. This is the script for Hindi which is the official language of India. It is also the script for Sanskrit, Marathi and Nepali languages. The script is used by more than 450 million people on the globe. A brief introduction of the script is given next from the OCR point of view.

A.1 Constituent Characters and Symbols of Devanagari

Devanagari alphabet, as used in Hindi, has eleven vowels and thirty three simple consonants. The vowels are shown in figure 71(a). Every vowel except the first one (see figure 71(a)) has a corresponding modifier form which is used to modify a consonant. The vowel modifier forms are shown in figure 71(b). The modifier symbol corresponding to 'आ' is placed next to the consonant. The modifiers corresponding to vowels 'ई, ओ' and 'औ' are represented by modifier symbol ' ' which is written next to the consonant and a top modifier which is written on top of the consonant

and modifier symbol ' ि'. The modifiers corresponding to vowels 'इ' is represented by modifier symbol ' ि' which is written before the consonant and a top modifier which is written on top of the consonant and modifier symbol ' ि'. All other modifier symbols are placed either at the top or at the bottom of a consonant. The placement of the modifier symbols is made clear by showing character 'क' with the modifier symbols in figure 71(b). Simple consonants are shown in figure 71(c). The consonants are compounded by writing one after the other, omitting in all but the last vertical stroke, and uniting the remainder of the consonant to the next following. The form, after omitting the vertical stroke, is referred to as pure derived form of the consonant. A *halant* ् is attached at the bottom of the consonant to represent the pure form of the consonants which do not have a vertical stroke at the end. The consonants in their pure forms are shown in figure 71(d). Consonant 'र' takes two different forms, according as it is the first or last letter of a compound. When initial in a conjunct, it is written as a top modifier of the second consonant as in 'र्'. But when non-initial, it takes the form of a short stroke below the preceding consonants (in 19 out of 33 consonants). This form of the consonants is referred to as 'rakar' form and the modifier as 'rakar' modifier. For the other 10 consonants, 'र' takes the form of a lower modifier. However, the remaining three consonants: 'क, त, श', change their form (see figure 71(e)). क्ष, ज्ञ are two compound consonants which are in common use. These compound consonants look different from their constituent consonants. *Anunasik* (ँ) and *anuswar* (ं) symbols denote the nasalization of a preceding vowel. These symbols are written directly over, or to the right of the vowel nasalized. Consonants ङ, ढ are often written with a diacritical point as ङ़, ढ़. *Visarg* (ः) symbol is used to indicate a weak aspiration. It is commonly omitted in Hindi and rarely used. These symbol are shown in in figure 71(g).

(a) Vowels

अ आ इ ई उ ऊ ऋ ए ऐ ओ औ

(b) Modifier Symbols corresponding to the vowels
(the modifier symbol has also been attached to
the consonant क to indicate its placing)

ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ
का	कि	की	कु	कू	कृ	के	कै	को	कौ

(c) Consonants

क	ख	ग	घ	ङ
च	छ	ज	झ	ञ
ट	ठ	ड	ढ	ण
त	थ	द	ध	न
प	फ	ब	भ	म
य	र	ल	व	श	ष	स	ह

(d) Pure Consonants

क्	ख्	ग्	घ्	ङ्
च्	छ्	ज्	झ्	ञ्
ट्	ठ्	ड्	ढ्	ण्
त्	थ्	द्	ध्	न्
प्	फ्	ब्	भ्	म्
य्	र्	ल्	व्	श्	ष्	स्	ह्

(e) Consonants with Rakar Modifier

क्र	ख्र	ग्र	घ्र	ङ्र
च्र	छ्र	ज्र	झ्र	ञ्र
ट्र	ठ्र	ड्र	ढ्र	ण्र
त्र	थ्र	द्र	ध्र	न्र
प्र	फ्र	ब्र	भ्र	म्र
य्र	र्र	ल्र	व्र	श्र	ष्र	स्र	ह्र

(f) Common Conjuncts

क्ष ज्ञ

(g) Additional Symbols

ं	ँ	़	ः
अं	अँ	ड़	दः

Figure 71: Characters and Symbols of Devanagari Script.

A.2 Composition of Characters and Symbols for Writing Words

A horizontal line is drawn on top of all characters of a word which is referred as header line or *shirorekha*. A character is usually written such that it is vertically separate from its neighbours. Whereas, a consonant in derived pure form is written such that it touches the following character. This results in a composite character. One or more modifier symbols and diacritic marks can be attached to a character.

It is convenient to visualize a Devanagari word in terms of three strips: a core strip, a top strip and a bottom strip. Top and bottom strips have only modifiers and diacritic marks whereas the core strip has the characters and vowel modifier ' ा'. Figure 72(a) shows a sentence and figure 72(b,c) and (d) show the contents of the three strips. The top and bottom strip may be empty for a word, only the top may

(a) An example sentence:
पुत्र, सुबह उठकर टहलने जाया करो
(b) Contents of the top strip:
ं ं
(c) Contents of the core strip:
पत्र, सबह उठकर टहलन जाया करा
(d) Contents of the bottom strip:
ु ु

Figure 72: An example showing the three strips of Devanagari words: (a) An example sentence; (b) Contents of the top strip; (c) Contents of the core strip; (d) Contents of the bottom strip.

be present or just the bottom. The core and top strip are separated by shirorekha. But no corresponding feature separates the lower strip from the core strip.